DATABASE ADMINISTRATION DB2 STANDARDS

Overview

This section describes steps necessary to evolve a logical data model into a physical schema and ultimately to a set of physical database structures in DB2. At CMS, tools have been identified to facilitate this process. CA/Platinum ERwin should be used to translate the approved logical data model to a physical model and BMC Change Manager should be used to implement data definitions generated from ERwin to DB2.

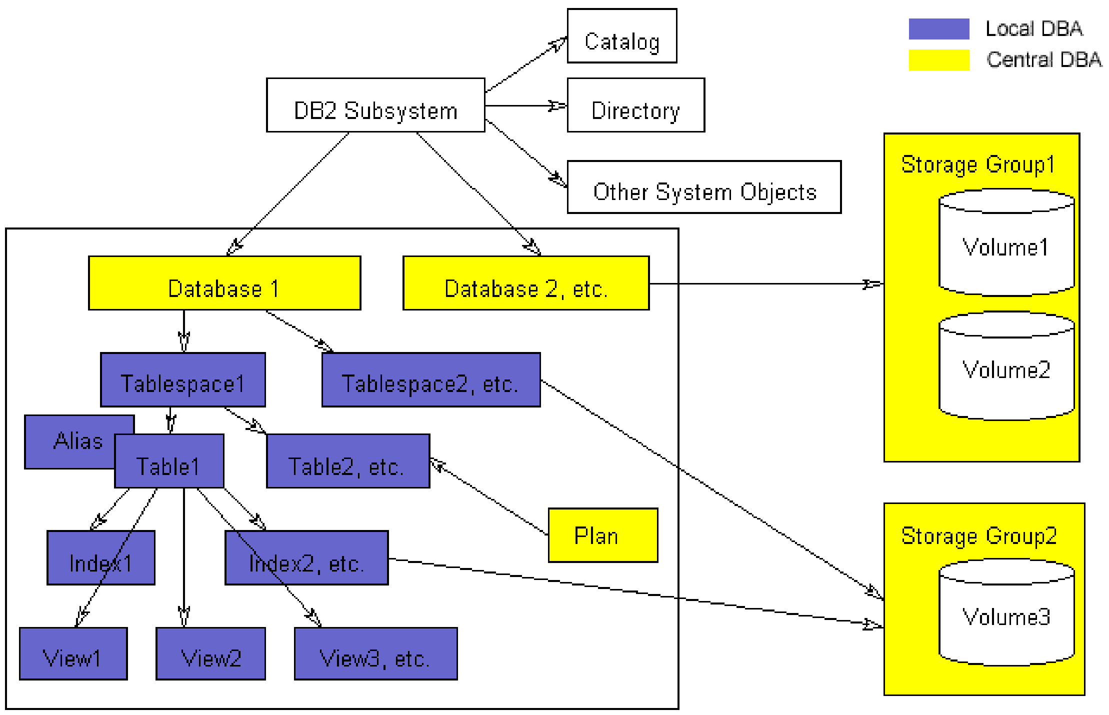
While developing the physical database design all standards must be followed, it is imperative that care and consideration be given to the standards noted throughout this DB2 Standards and Procedures document with regard to database object naming conventions, appropriate database object usage, and required object parameter settings. Although exceptions to the standards may be permitted, any deviation from the standards must be reviewed with and approved by the Central DBA staff prior to implementing into development.

DB2 Database Design Standards

1.1 DB2 Design Overview

Creating and maintaining objects in DB2 is a shared responsibility. Systems Programmers are responsible for all system related objects including the catalogs, directories, buffer pools, and other DB2 system resources. In the test subsystems, Central DBAs create and maintain objects in preparation for subsequent use by application development teams. These objects include storage groups, databases, and application plans. In the test environments, the Local DBAs are responsible for all database objects that will be utilized by their corresponding application. Tablespaces, tables, aliases, indexes, and views are all created and maintained by the local DBA

The following topics describe the standards Central and Local DBAs must follow when implementing physical database objects in DB2. This information must be applied even when the physical data model is developed and implemented using case tools, such as ERwin and BMC.



1.2 Databases

A Database in DB2 is an object which is created to logically group other DB2 objects such as tablespaces, tables, and indexes, within the DB2 catalog. Databases assist with the administration of DB2 objects by allowing for many operational functions to occur at the database level. DB2 commands such as -*START DATABASE* and -*STOP DATABASE*, for example, allow entire groups of DB2 objects to be started and stopped using one command.

See *Standard Naming Convention*.

1.2.1 Object Usage

STANDARD

Please refer to the roles and responsibilities documentation.

1.2.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining an application database. DB2 default settings must not be assumed for any of the following.

BUFFERPOOL bpname	Provide a valid bufferpool designation for tablespaces. The default value BP0 is used for the DB2 Catalog only Refer to the bufferpool section of the doc.
STOGROUP stogroup name	Specify a valid storage group. The storage group specified will indicate the default storage group that will be assigned to any tablespace or indexspace in the database ***only*** when that object was created without a storage group specification. ***Do not*** specify the default DB2 storage group (SYSDEFLT). ***Unauthorized tablespaces residing in the SYSDEFAULT storage group will be dropped without warning.*** *See Standard Naming Convention.*
INDEXBP bpname	Provide a valid bufferpool designation for indexes. The default value BP0 is used for the DB2 Catalog only. Refer to the bufferpool section of the doc.
CCSID encoding scheme	Specify the default encoding scheme, typically this will be EBCDIC.

1.3 Tablespaces

A *tablespace* is the DB2 object that holds data for one or more DB2 tables. It is a physical object that is managed in DB2.

See *Standard Naming Convention*.

1.3.1 Object Usage

STANDARD

Assign one table per tablespace. An exception to this standard is for tables which are primarily read-only edit/lookup tables containing a small number of rows (actual table size is less than 720K).

Create tablespaces explicitly using the *CREATE TABLESPACE* command rather than implicitly by creating a table without specifying a tablespace name.

Use DB2 storage groups to manage storage allocation for all application tablespaces.

Simple tablespaces are not supported at CMS.

Specify *CLOSE YES* for all tablespaces.

Specify *LOCKSIZE ANY* for all tablespaces that are not identified as 'read-only'. An exception to this standard is when it is determined that all updates to the table are performed in batch by a single job. In this case, *LOCKSIZE* table is recommended. Due to the additional overhead associated with maintaining row-level locks, use of *LOCKSIZE ROW* is permitted when a high level of application concurrency is necessary. Additionally, the Central DBA staff must review and approve the use of any *LOCKSIZE* option other than *LOCKSIZE ANY*.

Do not use the *LARGE* Parameter. Instead use the *DSSIZE* parameter for large tablespaces.

For read-only tables, define tablespaces with *PCTFREE, FREEPAGE of zero (0)*.

Specify *PRIQTY* and *SECQTY* quantities that fall on a track or cylinder boundary. On a 3390 device a track equates to 48K and a cylinder equates to 720K.

For tables over 100 cylinders always specify a *PRIQTY* and *SECQTY* to a number that is a multiple of 720.

Only specify *COMPRESS YES* when it has been determined that significant storage savings can be achieved by doing so. Typically, tablespaces containing primarily character data will benefit more than those containing mainly numeric data. Since data is compressed horizontally, tablespaces with longer average row lengths are also good candidates for compression. When determining the appropriateness of using DB2 compression, it is advisable to weigh the associated CPU overhead (especially when modifying data) against the savings in storage utilization as well as elapsed time for long running queries.

Partitioned Tablespace

When it is anticipated that a partitioned tablespace will hold more than 64 GB of data or a single partition will contain more than 1 GB of data, either now or in the future, include the *DSSIZE* parameter in the *CREATE TABLESPACE* command.

Define Partitioned tablespaces when it is anticipated that the number of rows in tablespace will exceed one million rows, two gigabytes of storage or when it is determined that partitioning the tablespace will yield performance benefit (i.e. parallel query processing or independent partition utility processing.).

For partitioned tablespaces, specify *LOCKPART YES* to encourage DB2 to invoke selective partition locking.

Segmented Tablespace

Define *SEGSIZE* as an even multiple of four (4), where the number chosen is closest to the actual number of pages on a table stored in the tablespace.

Specify *SEGSIZE* 64 for all tablespaces consisting of greater than 64 pages of data.

For segmented tablespaces, be sure that *PRIQTY* and *SECQTY* allocations are not less than the size of one segment. For example, if *SEGSIZE* is 64, both *PRIQTY* and *SECQTY* should be no less than 256K (64 pages * 4K/page).

1.3.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining an application tablespace. DB2 default settings must not be assumed for any of the following.

IN database name	Specified database must be a valid application database. Application tablespaces must not be defined in the DB2 default database. ***Unauthorized tablespaces residing in the DB2 default database, DSNDB04, will be dropped without warning***.
SEGSIZE	When the tablespace is not a partitioned tablespace, specify the size of each segment of pages within the tablespace. For most tables, this value should be set to 64 This parameter is ***mutually exclusive*** with NUMPARTS.
NUMPARTS integer	When the tablespace is to be partitioned, specify the number of partitions (datasets) that will comprise the entire tablespace. This parameter is ***mutually exclusive*** with *SEGSIZE*.
USING STOGROUP stogroup name	Indicate the DB2 storage group on which the application tablespaces will reside. ***Do not*** specify the default DB2 storage group (SYSDEFLT). ***Unauthorized tablespaces residing in the SYSDEFAULT storage group will be dropped without warning***. *See Standard Naming Convention.*
PRIQTY	Primary quantity is specified in units of 1K bytes. Specify a primary quantity that will accommodate all of the data in the tablespace or partition.
SECQTY	Secondary quantity is specified in units of 1K bytes. Specify a secondary quantity that is consistent with the anticipated growth of the table. The value specified should be large enough to prevent the tablespace from spanning more that three extents prior to the next scheduled *REORG.*
FREEPAGE	Indicate the frequency in which DB2 should reserve a page of free space on the tablespace or partition when it is reorganized or when data is initially loaded. Note: The maximum value for segmented tables is 1 less than the value specified for *SEGSIZE*.
PCTFREE	Indicate what percentage of each page on the tablespace or partition should be remain unused when it is reorganized or when data is initially loaded.

COMPRESS	Indicate whether DB2 should store data in the tablespace or partition in a compressed format
LOCKPART YES	For partitioned tablespaces, specify *LOCKPART YES* to encourage DB2 to invoke selective partition locking (SPL).
BUFFERPOOL bpname	Provide a valid bufferpool designation. ***Do not*** specify the default bufferpool of BP0, consult the bufferpool standards or the Central DBA staff.
LOCKSIZE	Specify the size of lock DB2 will acquire when data on the tablespace is accessed (see *Object Usage* standard above).
CLOSE	Indicate whether DB2 should close the corresponding VSAM dataset when no activity on tablespace is detected (see *Object Usage* standard above).
DSSIZE	Indicates the maximum data set size for each partition. A value in gigabytes that indicates the maximum size for each partition or, for *LOB* table spaces each data set. If you specify *DSSIZE* you must also specify *NUMPARTS* or *LOB*.

1.4 Tables

A table in DB2 is a named collection of columns and rows. The data represented in these rows can be accessed using SQL data manipulation language (select, insert, update, and delete). Generally, it is a table in DB2 that applications and end-users access to retrieve and manage business data.

See Standard Naming Convention.

1.4.1 Object Usage

STANDARD

Assign one table per tablespace. An exception to this standard is for tables which are primarily read-only edit/lookup tables containing a small number of rows (actual table size is less than 720K).

- Default tablespaces and databases must not be used. Tables must be created in a tablespace which was explicitly created for the corresponding application.
- In conformance with relational theory, all rows of a table should be uniquely identified by a column or set of columns to avoid the advent of duplicate rows. It is therefore required that every table defined to DB2 includes a primary key. If a table does not have a viable group of columns that can be defined as a primary key, consult the Central DBA staff for direction.

1.4.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining an application table. DB2 default settings must not be assumed for any of the following.

(*column definition*, ...)	List column specifications for each column that is to be defined to the table (see *Additional Table Definition Parameters* below for more details).
IN database.tablespace	Specify the fully qualified tablespace name where data for the table should be stored. ***Do not*** reference the DB2 default database. ***Unauthorized tablespaces residing in the DB2 Default database, DSNDB04, will be dropped without warning***.

1.5 Columns

Columns in a DB2 table contain data that was loaded, inserted or updated by some process. All columns have a corresponding datatype to indicate the format of the data within. In addition to datatypes, other edits can be associated with columns in DB2 in order to enforce defined business rule. These rules include default values, check constraints, unique constraints, and referential integrity (foreign keys).

See Standard Naming Convention.

1.5.1 Object Usage

STANDARD

- Nulls should only be used when there is a need to know the difference between the absence of a value versus a default value.
- The Central DA team should be consulted for a list of standard domains and column definitions.
- Columns that represent the same data but are stored on different tables; must have the same name, datatype and length specification. An example of this is where tables are related referentially. If on a PROVIDER table, for example, the primary key is defined as PROV_NUM CHAR(08), then dependent tables must include PROV_NUM CHAR(08) as a foreign key column.
- Columns that contain data which is determined to have the same domain must be defined using identical datatype and length specifications. For example, columns LAST_CHG_USER_ID and CASE_ADMN_USER_ID serve

different business purposes, however both should be defined as CHAR (08) in DB2. This standard applies to the entire enterprise and should not be enforced solely at an application level.

- All columns containing date information must be defined using the DATE data type.
- All columns containing time information must be defined using the TIME data type.
- Specify a datatype that most represents the data the column will contain. For numeric data, use one of the supported numeric data types taking into account the minimum and maximum value limits as well as storage requirements for each.
- For character data which may exceed 30 characters in length, consider use of the VARCHAR datatype. This could provide substantial savings in storage requirements. When weighing the benefits of defining a column to be variable in length, consider the average length of data that will be stored in this column. If the average length is less than 80% of the total column width, a variable length column *may* be appropriate. If data compression is used on the tablespace the central DBA should be contacted to determine if VARCHAR, should still be used.
- Consider sequence of the column definitions to improve database performance. Use the following as a guideline for sequencing columns on DB2 tables.

    - Primary Key columns (for reference purposes only)
    - Frequently read columns
    - Infrequently read columns
    - Infrequently updated columns
    - Variable length columns
    - Frequently updated columns

- For columns defined as DECIMAL be sure to use an odd number as the precision specification to ensure efficient storage utilization. Precision represents the entire length of the column, so in the definition DECIMAL (9,2), the precision is 9.
- For columns with whole numbers *SMALLINT* and *INTEGER* should be used.

1.5.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining columns in an application table. DB2 default settings must not be assumed for any of the following.

Column name	Provide a name for the column which conforms to DB2 Standard Naming Conventions.
datatype specification	Provide a datatype specification consistent with the characteristics of the data the column will hold.
NOT NULL/ NOT NULL WITH DEFAULT [default value]	Indicate that a value or default value is required for the column. Under some circumstances, this clause may not be required (see *Object Usage Standard* above).

- For additional information see IBM's Reference manuals.

1.6 Referential Constraints (Foreign Keys)

A referential constraint is a rule that defines the relationship between two tables. It is implemented by creating a foreign key on a dependent table that relates to the primary key (or unique constraint) of a parent table.

See Standard Naming Convention.

1.6.1 Object Usage

STANDARD

- Names for all foreign key constraints must be explicitly defined using data definition language (DDL). Do not allow DB2 to generate default names.
- DDL syntax allows for foreign key constraints to be defined along side of the corresponding column definition, as a separate clause at the end of the table definition, or in an ALTER table statement. Each of these methods are acceptable at CMS provided all of the required parameters noted below are included in the definition.
- Avoid establishing referential constraints for read-only tables.
- Do not define referential constraints on tables residing in multi-table tablespaces where the tables in the tablespace are not part of the same referential structure.
- When possible, limit the number of levels in a referential structure (all tables which have a relationship to one another) to three.

1.6.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining a foreign key constraint in an application table. DB2 default settings must not be assumed for any of the following.

constraint name	Specify a name for the foreign key constraint based on the see *Standard Naming Convention.*
(column name...)	Indicate the column(s) that make up the foreign key constraint. These columns must correspond to a primary key or unique constraint in the parent table.
REFERENCES table name	Specify the name of the table to which the foreign key constraint refers. If the foreign key constraint is based on a unique constraint of a parent table, also provide the column names that make up the corresponding unique constraint.
ON DELETE delete rule	Specify the appropriate delete rule which should be applied whenever an attempt is made to delete a corresponding parent row. Valid values include *RESTRICT*, *NO ACTION*, *SET NULL*, and *CASCADE*. (***Warning:*** *Use of the CASCADE delete rule can result in the mass deletion of numerous rows from dependent tables when one row is deleted from the corresponding parent. No response is returned to the deleting application indicating the mass delete occurred. Therefore, strong consideration should be given as to the appropriateness of implementing this rule in the physical design.)*

* For additional information see IBM's Reference manuals.

1.7 Table Check Constraints

A check constraint, (also known as a table or column constraint), is a rule defined to a table which dictates how edits will be applied against data that is either inserted or updated. The constraint rule is implemented using data definition language (DDL) and can apply to a specific column or multiple columns on a table. Constraint rules are always checked against one row of data at any point in time.

See Standard Naming Convention.

1.7.1 Object Usage

STANDARD

- Names for all check constraints must be explicitly defined using data definition language (DDL). Do not allow DB2 to generate default names.
- DDL syntax allows for check constraints to be defined along side of the corresponding column definition, as a separate clause at the end of the table definition, or in an ALTER table statement. Each of these methods are acceptable at CMS provided all of the required parameters noted below are included in the definition.

- Several restrictions apply when defining check constraints on a table or column. Refer to the *DB2 SQL Reference* manual for more details.
- Use caution when defining check constraints. Although DB2 will verify the syntax of SQL commands, it will not verify the meaning. It is possible to implement a check constraint that conflicts with other rules defined for the table. For example, the syntax listed below would be accepted by DB2 even though the column could never be inserted with a default value of 5.

```
    CREATE table
    DPP_TAB_A

(COL_1 SMALLINT NOT NULL WITH DEFAULT 5

        CONSTRAINT DPP013_COL1EDIT CHECK (COL_1 BETWEEN 10
        AND 20))
```

- Verify that check constraint rules do not conflict with any defined referential rules. For example, if a foreign key is defined as ON DELETE SET NULL and a check constraint is defined on the foreign key column as a rule that would prohibit nulls, DB2 will allow both rules to exist. However, any attempt to delete a parent row will fail due to the check constraint on the dependent table.

1.7.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining a table check constraint in an application table. DB2 default settings must not be assumed for any of the following.

CONSTRAINT constraint name	Specify a name for the check constraint based on the *Standard Naming Convention*.
CHECK (condition)	Indicate the business rule using valid SQL syntax. This rule will look similar to conditions expressed in an SQL *WHERE* clause.

* For additional information see IBM's Reference manuals.

1.8 Unique Constraints

A unique constraint is a rule defined to a table which indicates that any occurrence (row) in the table contains a distinct value in the column or combination of columns that make up the constraint. The constraint rule is implemented using data definition language (DDL) and can apply to a specific column or multiple columns on a table. A unique constraint is similar to a primary key where both implement entity integrity rules which state that each row in a table is uniquely identified by a non-null value or set of values. The major difference between primary keys and unique constraints is the fact that although only one primary key can be defined for one table, multiple unique constraints can exist.

1.8.1 Object Usage

STANDARD

- Names for all unique constraints must be explicitly defined using data definition language (DDL). Do not allow DB2 to generate default names.
- DDL syntax allows for unique constraints to be defined along side of the corresponding column definition, as a separate clause at the end of the table definition, or in an ALTER table statement. Each of these methods are acceptable at CMS provided all of the required parameters noted below are included in the definition.
- DB2 requires that a unique index be created to support each unique constraint defined for a table. DB2 will mark a table as unusable until such an index is created.

1.8.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining a unique constraint in an application table.

Constraint-name	Specify a name for the unique constraint based on the Standard Naming Convention.
(column name, ...)	Specify the name of the column or list of columns which make up the unique constraint.

1.9 ROWID

ROWID is a column data type that generates a unique value for each row in a table. Using the ROWID column as the partitioning key may allow for random distribution across partitions. DB2 will always generate the value for a ROWID column if "GENERATED ALWAYS" is specified. ROWID has an internal representation of 19 bytes which never changes. The first two bytes are the length field followed by 17 bytes for the ROWID. The external representation of the ROWID is 40 bytes with the RID appended(this changes with a REORG). ROWID values can only be assigned to a ROWID column or ROWID variable. When using a host variable to receive a ROWID it is necessary to declare the host variable as a ROWID for processing by the precompiler:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.

01 ABC-ROWID SQL TYPE IS ROWID,

EXEC SQL END DECLARE SECTION END-EXEC.
```

Assigning a character string to a ROWID you must first cast it to the ROWID data type. Two ways in which to cast a CHAR, VARCHAR, or HEX literal

- CAST (expression AS ROWID)
- CAST (X'hex_literal' AS ROWID)

1.9.1 Object Usage

STANDARD

- Table cannot have more than one ROWID column.
- Trigger cannot modify a ROWID column.
- ROWID column cannot be declared a primary or foreign key.
- ROWID column cannot be updated.
- ROWID column cannot be defined as nullable.
- Cannot have a ROWID column on a temporary table.
- ROWID column cannot have field procedures.
- ROWID column cannot have check constraints.
- Tables with a ROWID column cannot have an EDITPROC
- ROWID data type cannot be used with DB2 private protocol, can only be used with DRDA.
- ROWID column is required for implementing LOBs.

1.9.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining a ROWID column. DB2 default settings must not be assumed for any of the following.

GENERATED BY DEFAULT	DB2 accepts valid row ID's as inserted values for a row. DB2 will generate a value if none are specified. Must define a unique single column index on the ROWID column (cannot INSERT until created). Recommended for tables populated from another table.
GENERATED ALWAYS	DB2 will always generate value. An index is not required if generated with this approach. LOAD utility cannot be used to load ROWID values if the ROWID column was created with the GENERATED ALWAYS clause. This is the recommended approach.

* For additional information see IBM's Reference manuals.

1.10 Identity Columns

A column type for generating unique sequential value for a column when a row is added to the table. Sequential incrementing is not guaranteed if more than one transaction is updating the table. For a unit of work that is rolled back, the allocated numbers that have already been used are not reissued.

1.10.1 Object Usage

STANDARD

- Can have only one Identity column per table.
- Identity column do not allow nulls.
- Cannot have a FIELDPROC.
- Table(s) with an Identity column cannot have an EDITPROC.
- Use of "WITH DEFAULT" is not allowed.
- The data type of an identity column can be INTEGER, SMALLINT, or DECIMAL with a scale of 0. The column can also be a distinct type based on one of these data types.
- Issuing a recovery to a prior point in time will cause DB2 not to reissue those values already used.
- CREATE table … LIKE … INCLUDING IDENTITY causes the newly created table to inherit the identity column of the old table.
- The ALTER table statement can be used to add an identity column. If the table is populated when the ALTER is issued, the table is placed in the REORG pending state.

1.10.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining an Identity column. DB2 default settings must not be assumed for any of the following.

GENERATED BY DEFAULT	An explicit value can be provided, if a value is not provided then DB2 generates a value. Uniqueness is not guaranteed except for previously DB2 generated values. You can create an unique index to guarantee unique values. If this is done, developers need to check for -803 SQLCODE indicating an attempt to insert a duplicate value in a column with an unique index.
GENERATED ALWAYS	DB2 will always generate a value, an explicit value cannot be specified with an INSERT and UPDATE statement
AS IDENTITY	This designates the column as an identity column. The column is implicitly NOT NULL.
START WITH	Only need to specify if the number is not "1" furthermore, must be a valid number for the chosen data type. This number can be a positive or negative number. If it is a positive number, it is an ascending number. If it is a negative number, it is a descending number.
INCREMENTED BY	Value by which the Identity column will be incremented (1 is the default). The value must be valid for the chosen data type.
CACHE 20	Preallocated values that are kept in memory by DB2 (the default is 20). The minimum value that can be specified is 2, and the maximum is the largest value that can be represented as an integer. During a system failure, all cached Identity column values that are yet to be assigned are lost, and thus, will never be used. Therefore, the value specified for CACHE also represents the maximum number of values for the identity column that could be lost during a system failure.
NO CACHE	No preallocated values are kept in memory by DB2. This is useful if it is necessary to assign numbers in sequence as the rows are

	inserted in a sysplex data sharing environment.

* For additional information see IBM's Reference manuals.

1.11 Views

A View in DB2 is an alternative representation of data from one or more tables or views. Although they can be accessed using data manipulation language (*SELECT, INSERT, DELETE, UPDATE*), views do not contain data. Actual data is stored with the underlying base table(s). Views are generally created to solve business requirements related to security or ease of data access. A view may be required to limit the columns or rows of a particular table a class of business users are permitted to see. Views are also created to simplify user access to data by resolving complicated SQL calls in the view definition.

See Standard Naming Convention.

1.11.1 Object Usage

STANDARD

View definitions must reference base tables only. Do not create views which reference other views. Use views sparingly. Create views only when it is determined that direct access to the actual table does not adequately serve a particular business need. In general, views should not be required if the data on a table is not sensitive and is only accessed using predefined SQL such as static embedded SQL.

1.11.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining a view. DB2 default settings must not be assumed for any of the following.

(*column list*, …)	List column names for each column that is to be defined to the view. These names must conform to the DB2 Standard Naming Convention for column names.
AS select statement	Specify SQL select statement that defines this view. Do not include existing views as part of the view definition. ***Do not include a SELECT **** … as a select statement.

1.12 Indexes

An index is an ordered set of data values with corresponding record identifiers (*RID*s) which identify the location of data rows on a tablespace. An index is created for one table and consists of data values from one or many columns on the table. When created, one or more VSAM linear datasets used to hold the data and RID values is also created. This physical object (dataset), known as an indexspace, can be managed in DB2 using storage groups (DB2 Managed).

See Standard Naming Convention.

1.12.1 Object Usage

STANDARD

- Use DB2 storage groups to manage storage allocation for all indexes.
- Specify *CLOSE YES* for all indexes.
- Specify *COPY YES* for all indexes that will be Image Copied.
- For read-only tables, define indexes with *PCTFREE, FREEPAGE*.
- When creating an index on a table that already contains over 5,000,000 rows, use the *DEFER* option then use a *REBUILD INDEX* utility to build the index. By stating *DEFER YES* on the *CREATE INDEX* statement, DB2 will simply add the index definition to the DB2 catalog; the actual index entries will not be built. Using this method can significantly reduce the amount of resources needed to create an index on a large table.
- One index on every table must be explicitly defined as the clustering index.
- Specify *PRIQTY* and *SECQTY* quantities that fall on a track or cylinder boundary. On a 3390 device a track equates to 48K and a cylinder equates to 720K.
- For indices over 100 cylinders always specify a *PRIQTY* and *SECQTY* on a cylinder boundary, or a number that is a multiple of 720.

1.12.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining an index. DB2 default settings must not be assumed for any of the following.

CLUSTER	One index for each table must be designated as the *clustering index*. Not doing so will cause DB2 to assume the first index created on the table as the *clustering index.*
USING	Indicate the DB2 storage group on which the indexspace will

STOGROUP stogroup name	reside. ***Do not*** specify the default DB2 storage group (SYSDEFLT). ***Unauthorized objects defined in the default storage group will be dropped*** without warning. See *Standard Naming Convention*.
PRIQTY	Primary quantity is specified in units of 1K bytes. Specify a primary quantity that will accommodate all of the index entries in the table or partition (for partitioned indexes).
SECQTY	Secondary quantity is specified in units of 1K bytes. Specify a secondary quantity that is consistent with the anticipated growth of the table or partition (for partitioned indexes). The value specified should be large enough to prevent the indexspace from spanning more that three extents prior to the next scheduled REORG.
FREEPAGE	Indicate the frequency in which DB2 should reserve a page of free space on the indexspace when data is initially loaded to the table or when the index or index partition is rebuilt or reorganized.
PCTFREE	Indicate what percentage of each page on the indexspace should be remain unused when data is initially loaded to the table or when the index or index partition is rebuilt or reorganized.
BUFFERPOOL bpname	Provide a valid bufferpool designation. The default value is BP0 which should never be used - it is reserved for the DB2 catalog. Consult with the central DBA to determine the appropriate bufferpool setting for your database objects.
CLOSE YES	Indicate whether DB2 should close the corresponding VSAM dataset when no activity on indexspace is detected (see *Object Usage* standard above).
COPY	Indicates Whether the *COPY* utility is allowed for this index. Valid values are *NO* or *YES*.

1.13 Table Alias

A DB2 Alias provides an alternate name for a table or view that resides on the local or remote database server. At CMS, aliases will be maintained to allow application programs to reference unqualified table and view names.

See Standard Naming Convention.

1.13.1 Object Usage

STANDARD

Aliases are utilized in the development environments; see for details.

1.13.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining an alias. DB2 default settings must not be assumed for any of the following.

CREATE ALIAS alias name	Specify an alias name consistent with the *Standard Naming Convention*.
FOR owner.tablename	Provide a fully qualified table name (creator.tablename) for the corresponding table.

1.14 Synonyms

A DB2 Synonym is an alternate name an individual can assign to a table or a view. After creating a synonym, the individual can refer to the unqualified synonym name (name without a creator prefix). DB2 will recognize the unqualified name as a synonym and will translate the synonym name to the actual fully qualified table or view name (creator.name). With current releases of DB2, synonyms no longer provide a significant benefit in a development environment and will not be supported at CMS.

See *Standard Naming Convention.*

Synonyms are not supported at CMS. Standard naming conventions do not apply.

1.14.1 Object Usage

STANDARD

Applications developed and maintained at CMS must not reference DB2 synonyms. Synonyms will not be supported in the validation and production environments.

1.15 Stored Procedures

A Stored Procedure is a compiled program defined at a local or remote DB2 server that is invoked using the SQL CALL statement.

See Standard Naming Convention.

1.15.1 Object Usage

STANDARD

NOTE: Central DBA Approval Required
The use of Stored Procedures is closely governed by the Central DBA Group (CDBA). The CDBA Group must approve all Stored Procedures, as part of the Database and Pre-Validation Walkthroughs. The application is responsible for notifying the Central DBA of the intent to use Stored Procedures and what purpose the Stored Procedures will have, **prior to developing and testing**. This will include but is not limited to, business requirements, type of access, and performance considerations.

- Stored Procedures will be maintained, migrated and compiled through Endevor.
- Nested Stored Procedures are not permitted at CMS. After invoking the initial stored procedure subsequent procedures should be invoked using application language Call/Link statements. Nested Stored Procedure calls have been proven to be inefficient.
- Close and commit statements must be executed in the invoking modules to release held DB2 resources.
- Stored Procedures will be defined as *STAYRESIDENT YES*.
- Stored Procedures will be defined as *PROGRAM TYPE SUB*.
- All Stored Procedures will be defined with the *FENCED* parameter.
- All Stored Procedures will be defined to execute under the control of a Work Load Manager environment.

1.15.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining a Stored Procedure. *DB2 default settings must not be assumed for any of the following.*

IN, OUT, INOUT	Identifies the parameter as an input, output or input and output

	parameter to the Stored Procedure.
DYNAMIC RESULT SETS integer	Specifies the number of query result sets that can be returned. A value of zero indicates no result sets will be returned.
EXTERNAL NAME procedure-name	Identifies the user-written program/code that implements the Stored Procedure.
LANGUAGE	Identifies the application programming language that the Stored Procedure is coded in.
PARAMETER STYLE	Specifies the linkage convention used to pass parameters to the Stored Procedure.
FENCED	Specifies that the Stored Procedure runs in an external address space which prevents user programs from corrupting DB2 storage.
DBINFO	Specifies whether specific information known by DB2 is passed to the Stored Procedure when it is called.
COLLID	Identifies the package collection that is used when the Stored Procedure is called.
WLM ENVIRONMENT	Identifies the MVS Workload Manager (WLM)environment that the Stored Procedure is to run in.
STAY RESIDENT YES	Specifies whether the Stored Procedure load module stays resident in memory when the stored procedure terminates.
PROGRAM TYPE SUB	Identifies if the Stored Procedure runs as a Main or a Subroutine.
SECURITY	Identifies how the Stored Procedure interacts with RACF to control access to non-DB2 resources. Only USER and DB2 are allowed.
COMMIT ON RETURN	Specifies if DB2 commits the transaction immediately on return from the Stored Procedure.

1.16 User Defined Functions

A User Defined Function (or UDF) is similar to a host language subprogram or function that is invoked in an SQL statement.

1.16.1 Object Usage

UDF's are not yet a standard at CMS. If there is a need for a UDF, contact the Central DBA for guidance.

STANDARD

* For additional information see IBM's Reference manuals.

1.17 User Defined Types

A Distinct Type (or User Defined Data Type) is based on a built-in database data type, but is considered to be a separate and incompatible type for semantic purposes. Example: One could define a US_Dollar and Mexican Peso data types, although they may both be defined as DECIMAL(10,2), the business may want to prevent these columns from being compared to one another.

There are currently no written standards for UDTs. If there is a need for UDTs please contact the Central DBA group for guidance.

1.17.1 Object Usage

STANDARD

* For additional information see IBM's Reference manuals.

1.18 Triggers

A Trigger is a defined set of SQL defined for a table that executes when a specified SQL event occurs.

See Standard Naming Convention.

1.18.1 Object Usage

STANDARD

1.18.2 Required Parameters (DDL Syntax)

STANDARD

The parameters listed below must be included in the DB2 data definition language (DDL) when defining a Trigger. DB2 default settings must not be assumed for any of the following.

ON table table-name	Identifies the triggering table that the trigger is associated with.
FOR EACH	*ROW* - Specifies that DB2 executes the trigger for each row of the triggering table that the triggering SQL operation changes or Inserts. *STATEMENT* - Specifies that DB2 executes the triggered action only once for the triggering SQL operation.
WHEN	Identifies a condition that evaluates to true, false, or unknown.
BEGIN AUTOMIC	Specifies the SQL that is executed for the triggered action.

* For additional information see IBM's Reference manuals.

1.19 LOBs

A *large object* is a data type used by DB2 to manage unstructured data. DB2 provides three built-in data types for storing large objects:

Binary Large Objects, also known as *BLOBs* can contain up to 2GB of binary data. Typical uses for *BLOB* data include photographs and pictures, audio and sound clips, and video clips.

Character Large Objects, also known as *CLOBs* can contain up to 2GB of single byte character data. *CLOBs* are ideal for storing large text documents in a DB2 database.

Double Byte Character Large Objects, also known as *DBCLOBs* can contain up to 1GB of double byte character data, for a total of 2GB. *DBCLOBs* are useful for storing text documents in languages that require double byte characters, such as Kanji.

BLOBs, *CLOBs*, and *DBCLOBs* are collectively referred to as *LOBs*. The actual data storage limit for *BLOB*, *CLOBs*, and *DBCLOBs*, is 1 byte less than 2 gigabytes of data.

ROWID column is required for implementing *LOBs*.

There are currently no written standards for *LOBs*. If there is a need for *LOBs* please contact the Central DBA group for guidance.

1.19.1 Object Usage

STANDARD

* For additional information see IBM’s Reference manuals.

1.20 Buffer Pools

Buffer pools are areas of virtual storage where DB2 temporarily stores pages of table spaces or indexes. When an application program requests a row of a table, the page containing the row is retrieved by DB2 and placed in a buffer. If the requested page is already in a buffer, DB2 retrieves the page from the buffer, significantly reducing the cost of retrieving the page.

1.20.1 Object Usage

STANDARD

1.20.2 Required Parameters (DDL Syntax)

STANDARD

Bufferpools are assigned and created by the Software Support Group with Central DBA input.

The Bufferpool assignments are listed below:

BP1	Large Tables (> 48000K or > 1000 tracks)
BP2	Large Indexes (> 48000K or > 1000 tracks)
BP3	Medium Tables ((> 240K and < 48000K) or (> 5 tracks and < 1000 tracks))
BP4	Medium Indexes ((> 240K and < 48000K) or (> 5 tracks and < 1000 tracks))
BP5	Small Tables (< than 240K or < than 5 tracks)
BP6	Small Indexes (< than 240K or < than 5 tracks)

1.21 Space Requests

It is the responsibility of the Project GTL to request necessary space from the DASD team. The space required in each subsystem (all test environments and production) must be delivered as part of the pre-development walk-through.

1.22 Capacity Planning

It is the responsibility of the Project GTL to schedule a meeting with Lockheed Martin and the Central DBA team to discuss and plan for impacts a new system or major modifications to an existing system will have on the CMS computer environments. This includes but not limited to Development, Test, Performance Testing, Production, etc.

Application Programming

2.1 Data Access (SQL)

STANDARD

The following is a list of SQL usage standards to which all application programs developed and maintained at CMS must adhere.

General SQL Usage

- Application programs must not include data definition language (DDL) or data control language (DCL) SQL statements.
- SQL statements must include unqualified table and view names only. At no time should a table name be referenced from an application program using the table creator as a prefix.
- SQL columns must be accessed with elementary data items. At no time should host structures be referenced in an SQL statement. The use of group level data items, or structures are only permitted to manipulate variable length data or null indicators, (i.e., VARCHAR).

SQL Error Handling

- All application programs must check the value of SQLCODE immediately after each executable SQL command is issued, to determine the outcome of the SQL request. Depending on the requirements of the application program, appropriate logic to handle all possible values of SQLCODE must be performed
- All application programs must utilize a CMS Standard SQL Error Handling Routine calling DSNTIAR and formatting output to SYSOUT to handle all unexpected DB2 error conditions that are not accommodated within the application.
- Standard Error Handling Routines
- A complete list of SQLCODEs can be located in the DB2 Messages and Codes or DB2 Reference Summary manuals.

DECLARE TABLE

- Table Declarations for all tables and views accessed must be generated using the DB2 DCLGEN command.
- Table declarations must be stored individually as members in a Endevor library. Table declarations will be expanded in application source code at program preparation time using the EXEC SQL, INCLUDE command.

HOST VARIABLES

- Host Variable Declarations must be defined as elementary data items in the Working-Storage section in Cobol programs. Group level data items as host variables are only permitted to manipulate variable length data or null indicators, (i.e., VARCHAR).

DECLARE CURSOR

- DECLARE CURSOR statements must be defined in the Working-Storage section in Cobol programs.
- Since cursor definitions include a valid SQL SELECT statement, DECLARE CURSOR statements must also adhere to the SELECT statement standards.
- Cursor declarations for read-only cursors must include the FOR FETCH ONLY, to make use of block fetching (improved I/O).
- Cursor declarations for cursors which will retrieve a small number of rows, must include the OPTIMIZE FOR 1 ROWS clause to avoid sequential prefetch processing (increased I/O).

SELECT

- SELECT statements must include an explicit list of column names and expressions, which are being retrieved from DB2. Asterisks (*) are only permitted in SELECT statements under the following circumstances:
  - 1. When using the SQL COUNT column function to determine a number of rows (SELECT COUNT(*) FROM table-name...); or
  - 2. When coding a subselect to determine the existence of a condition (... WHERE EXISTS (SELECT * FROM table-name ...)).

INSERT

- INSERT statements must use the format of a column list with corresponding host variables for each column. INSERT INTO tablename (col1, col2, col3,...) VALUES (:hvar1, :hvar2, :hvar3...)

LOCK table

- Due to the adverse affect on concurrent usage of DB2 resources, the use of the LOCK table command is limited. Use of the LOCK table statement must be approved by the Central DBA staff when it is required to achieve improved performance or data integrity.

CLOSE CURSOR

- All application programs designed to process SQL cursors must execute a CLOSE CURSOR statement for every corresponding OPEN CURSOR.

2.2 Application Recovery

To achieve a high level of data recoverability and application concurrency, all DB2 application programs updating data, (INSERT, DELETE, UPDATE), will incorporate logical unit of work processing. Application logic will clearly identify begin and end points for sets of work which must be processed as a whole. Commit and rollback commands will be incorporated where appropriate.

Additionally, all batch DB2 application programs will include abend/restart logic which will minimize the impact to DB2 resources as well as the allotted batch processing window in the event of an application failure.

Quickstart/MVS from BMC is the standard development tool that must be used at CMS in order to comply with this standard.

2.3 Program Preparation

2.3.1 DB2 Package

All DB2 application programs developed and maintained at CMS are bound to DB2 as packages and are included in DB2 plans using the PKLIST parameter in the BIND PLAN statement. Application programs are normally bound as part of the Endevor procedures.

Additionally, all DB2 packages must be bound using an ISOLATION level of 'CS' (cursor stability). Specifying any other isolation level, such as repeatable read (RR) or uncommitted read (UR) requires approval of the Central DBA staff.

The following is an example of BIND PACKAGE syntax which should be used to create DB2 packages at CMS:

```
BIND PACKAGE (collection-id)    -
  OWNER(DEV$xxx)                -
  QUALIFIER(DBA$DBxx)           -
```

```
MEMBER(program-name)      -
RELEASE(COMMIT)           -
ISOLATION(CS)             -
ACTION(REPLACE)           -
EXPLAIN(YES)
```

2.3.2 DB2 Plan

All DB2 applications developed and maintained at CMS will implement DB2 plans which contain a list of DB2 packages only. Use of the MEMBER parameter in the BIND PLAN statement to include DBRM members is prohibited.

Additionally all DB2 plans must be bound using an ISOLATION level of 'CS' (cursor stability). Specifying any other isolation level, such as repeatable read (RR) or uncommitted read (UR) requires approval of the Central DBA staff.

2.3.3 Explain

One set of Plan tables will be created by the Central DBA group for each application development RACF group (ex DEV$NPS). Endevor will produce explain reports for Central DBA review as part of the Endevor migrations.

2.4 DB2 Development Tools

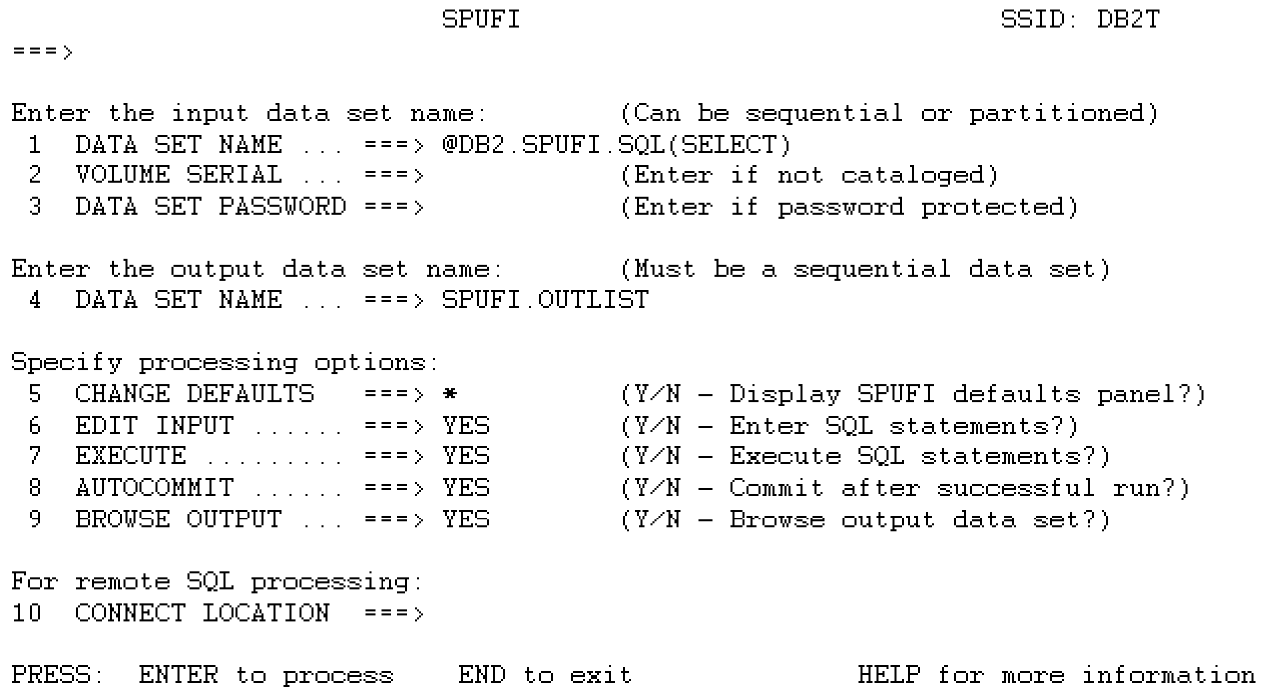
2.4.1 SPUFI (SQL Processing Using File Input)

A TSO facility, *SPUFI* (SQL Processing Using File Input), is provided with every installation of DB2 which allows users to execute SQL statements interactively. As it is named, SPUFI reads, as input, a dataset containing one or many SQL statements. SPUFI sends these statements to DB2 for execution and writes the results of the execution to another dataset which SPUFI then presents in a ISPF Browse session for inspection. SPUFI is a valuable development tool, in that it allows you to test SQL statements, create or verify test data, and review application data structure formats.

To use SPUFI, choose option D;I from the CMS *ISPF/PDF Primary Option Menu*. From the *DB2I Primary Option Menu*, choose option 1 for SPUFI.

Define Execution Environment:

Below is a sample of a SPUFI primary panel. It is used to control the execution flow of your SPUFI session. Fields 1 through 3 identify the name of the dataset which will contain the SQL script to be processed, (typically a member of a partitioned dataset). The name of the dataset where SPUFI should write the results of the SQL processing must be provided in field 4. The dataset specified in this field will be deleted and redefined each time SPUFI executes an SQL script. Fields 5 through 9 define the execution environment. Field 10 is used to indicate the remote DB2 location where the SQL should be processed. In most cases this field should remain blank. The purpose of each field is described in detail in Chapter 2-5 of the *DB2 Applications Programming and SQL Guide.*

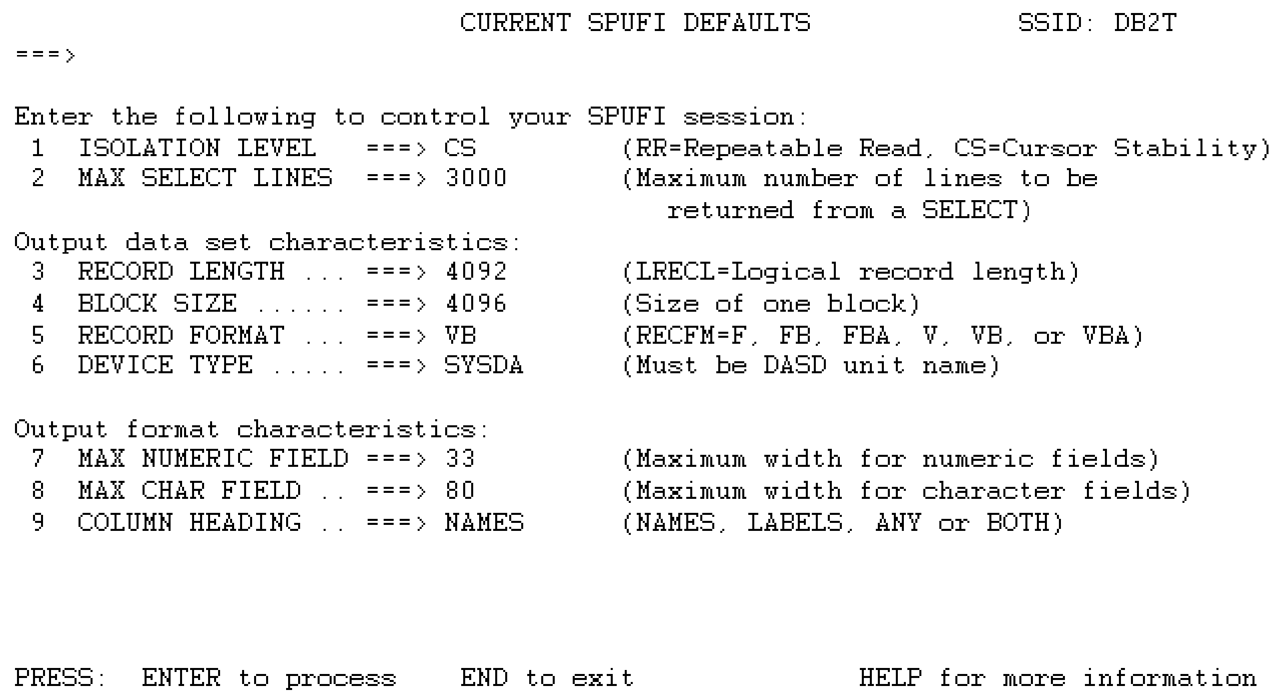


```
                           SPUFI                              SSID: DB2T
===>

Enter the input data set name:        (Can be sequential or partitioned)
 1  DATA SET NAME ... ===> @DB2.SPUFI.SQL(SELECT)
 2  VOLUME SERIAL ... ===>            (Enter if not cataloged)
 3  DATA SET PASSWORD ===>            (Enter if password protected)

Enter the output data set name:       (Must be a sequential data set)
 4  DATA SET NAME ... ===> SPUFI.OUTLIST

Specify processing options:
 5  CHANGE DEFAULTS   ===> *          (Y/N - Display SPUFI defaults panel?)
 6  EDIT INPUT ...... ===> YES        (Y/N - Enter SQL statements?)
 7  EXECUTE ......... ===> YES        (Y/N - Execute SQL statements?)
 8  AUTOCOMMIT ...... ===> YES        (Y/N - Commit after successful run?)
 9  BROWSE OUTPUT ... ===> YES        (Y/N - Browse output data set?)

For remote SQL processing:
10  CONNECT LOCATION  ===>

PRESS:  ENTER to process    END to exit             HELP for more information
```

To execute SQL using SPUFI, enter all required information in the SPUFI primary panel. At a minimum, specify the name of the dataset which will contain the SQL statements in field 1 and the name of the dataset SPUFI should allocate to write output results in field 4. Verity that the processing options in fields 5 through 9 are as desired and press the *Enter* key to continue.

Changing Defaults:

Depending on your needs, it is sometimes necessary to change the default settings for your SPUFI session. By placing a 'YES' in field 5 SPUFI primary panel, SPUFI will present the following *Defaults* panel once the *Enter* key has been pressed.



```
                           CURRENT SPUFI DEFAULTS            SSID: DB2T
===>

Enter the following to control your SPUFI session:
 1  ISOLATION LEVEL   ===> CS          (RR=Repeatable Read, CS=Cursor Stability)
 2  MAX SELECT LINES  ===> 3000        (Maximum number of lines to be
                                          returned from a SELECT)
Output data set characteristics:
 3  RECORD LENGTH ... ===> 4092        (LRECL=Logical record length)
 4  BLOCK SIZE ...... ===> 4096        (Size of one block)
 5  RECORD FORMAT ... ===> VB          (RECFM=F, FB, FBA, V, VB, or VBA)
 6  DEVICE TYPE ..... ===> SYSDA       (Must be DASD unit name)

Output format characteristics:
 7  MAX NUMERIC FIELD ===> 33          (Maximum width for numeric fields)
 8  MAX CHAR FIELD .. ===> 80          (Maximum width for character fields)
 9  COLUMN HEADING .. ===> NAMES       (NAMES, LABELS, ANY or BOTH)



PRESS:  ENTER to process    END to exit              HELP for more information
```

This panel is used to change SPUFI settings such as the allocation parameters for your output dataset and limiting parameters such as the maximum number of rows SPUFI should return from a single select statement and maximum field widths that should be displayed. Most importantly, this panel allows you to select an ISOLATION LEVEL for your SPUFI session. ISOLATION LEVEL indicates to DB2 at what duration should it retain a hold on data while you are reading it through SPUFI. The value you provide in this field has a major impact on the availability of the data you are reading to other DB2 users and applications.

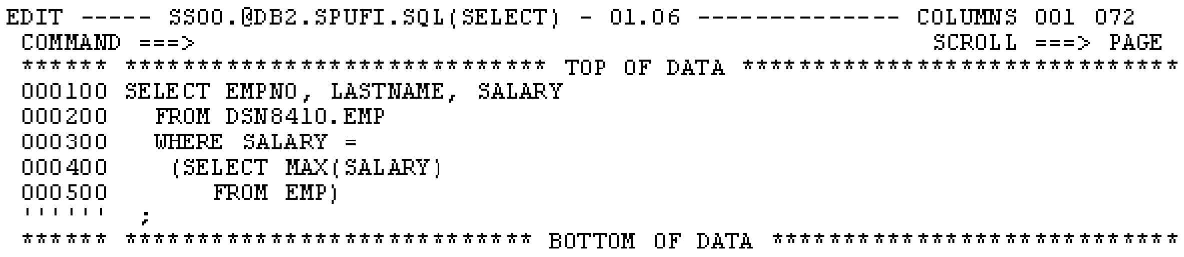
STANDARD

At CMS, ISOLATION LEVEL must be set to Cursor Stability (CS) for the duration of a SPUFI session to ensure the highest level of concurrent read access to DB2 data.

To modify the default settings for you SPUFI session, make the desired changes in the appropriate field on the *Current SPUFI Defaults* panel and press the *Enter* key.

***Input Dataset*:**

Prior to reading the input dataset specified on the SPUFI primary panel, SPUFI will give you an opportunity to edit this dataset by presenting it to you though an ISPF edit session (see example below). One or more SQL statements can be entered in the input dataset. Each statement must be delimited with a semicolon (;). Comments are also permitted in a SPUFI script and are indicated by a double-hyphen (--) at the beginning of the comment text.

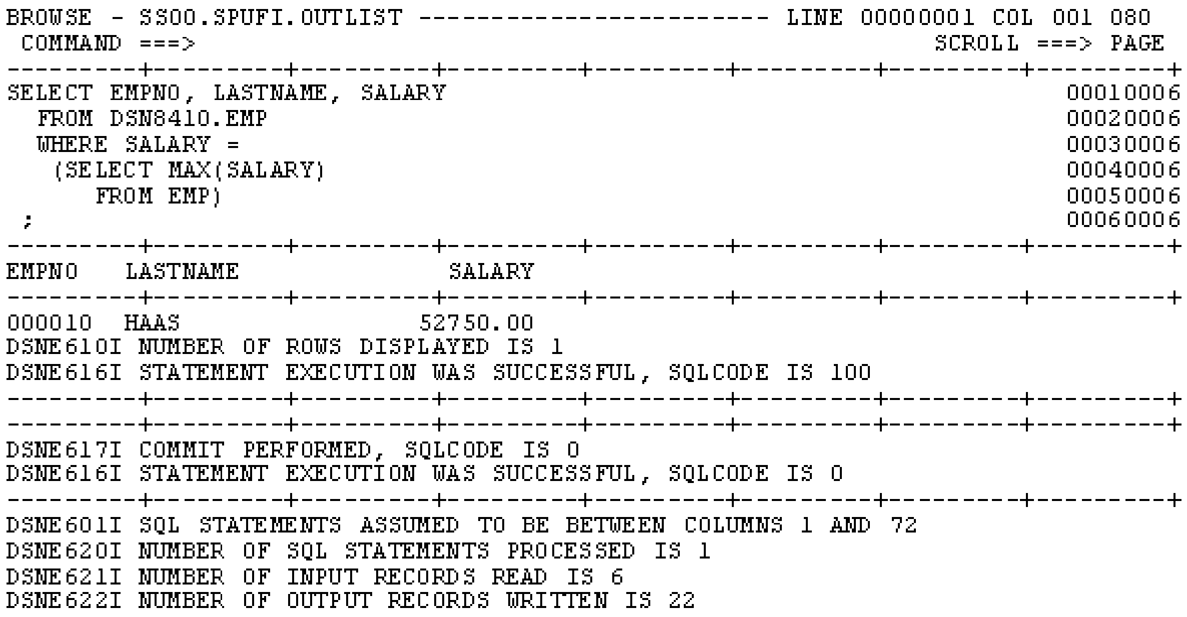


```
EDIT ----- SS00.@DB2.SPUFI.SQL(SELECT) - 01.06 -------------- COLUMNS 001 072
 COMMAND ===>                                                  SCROLL ===> PAGE
 ****** ***************************** TOP OF DATA ******************************
        SELECT EMPNO, LASTNAME, SALARY
          FROM DSN8410.EMP
          WHERE SALARY =
           (SELECT MAX(SALARY)
              FROM EMP)
 ''''''  ;
 ****** **************************** BOTTOM OF DATA ****************************
```

Once you are satisfied with the input SQL, press PF3 to save the data and to return to the SPUFI primary panel. Press enter again to execute the SQL script and to view the results.

SQL Results:

While executing the SQL script, SPUFI writes the results from DB2 to the output dataset specified in field (4) of the SPUFI primary panel. This dataset is presented in an ISPF browse session for you to review once SPUFI is complete. Below is an example output that would be presented by SPUFI.



```
BROWSE - SS00.SPUFI.OUTLIST ------------------------ LINE 00000001 COL 001 080
 COMMAND ===>                                                  SCROLL ===> PAGE
---------+---------+---------+---------+---------+---------+---------+---------+
SELECT EMPNO, LASTNAME, SALARY                                         00010006
  FROM DSN8410.EMP                                                     00020006
  WHERE SALARY =                                                       00030006
   (SELECT MAX(SALARY)                                                 00040006
      FROM EMP)                                                        00050006
 ;                                                                     00060006
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO   LASTNAME              SALARY
---------+---------+---------+---------+---------+---------+---------+---------+
000010  HAAS                52750.00
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------+
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 6
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 22
```

In addition to the expected output (i.e., result set from a SELECT statement), SPUFI also provides other information to indicate the status of the execution. When relevant, it will indicate the number of rows in a table that were affected by the previous SQL statement, the SQLCODE received from DB2 for each issue SQL statement, whether or not a COMMIT was performed, number of input records read from the SQL script, number of output records written to the output dataset, etc.

Caution: When evaluating the results of a SPUFI execution, be sure to review **all** of the results carefully. SPUFI will discontinue processing an SQL script the first time it encounters an error condition (i.e., SQLCODE < 0).

2.4.2 QMF (Query Management Facility)

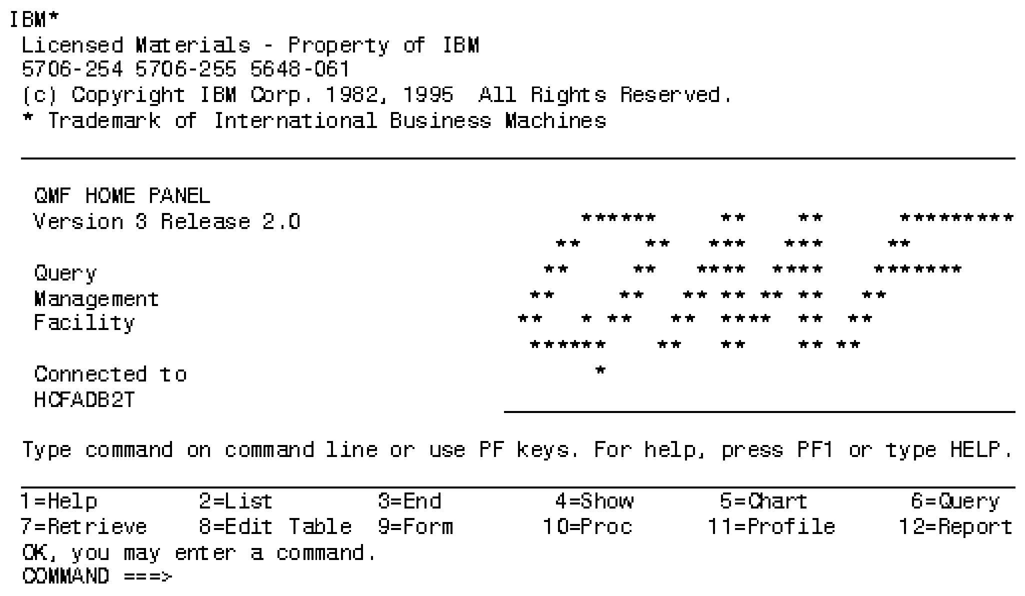
QMF is an ISPF facility which is available to simplify access to data stored in DB2. Typically, QMF is used to quickly generate formatted reports to fulfill adhoc requests. However it can also be used to develop elaborate reporting applications which run frequently in a production environment.

The following sections describe two methods of using QMF, both of which support interactive and batch activity. When using QMF to run adhoc queries, either method can be used. However, when it is anticipated that the QMF application you are developing will be migrated to production, the procedures noted under *QMF Application Development* must be used.

2.4.2.1 QMF Adhoc Usage

A facility is available at CMS which provides a quick and easy method to generate formatted reports based on information stored in DB2. This facility, QMF, is an ISPF application which can be run interactively and in batch. The following describes basic procedures to run a query and produce a report (in a default format) from an interactive QMF session. For more information on executing QMF in batch at CMS, see *QMF Application Development (QMF Import Facility)* section of this document. To learn more about QMF features including formatting reports, please refer to the *QMF Learner's Guide* and *QMF Advanced User's Guide*.

QMF Panels



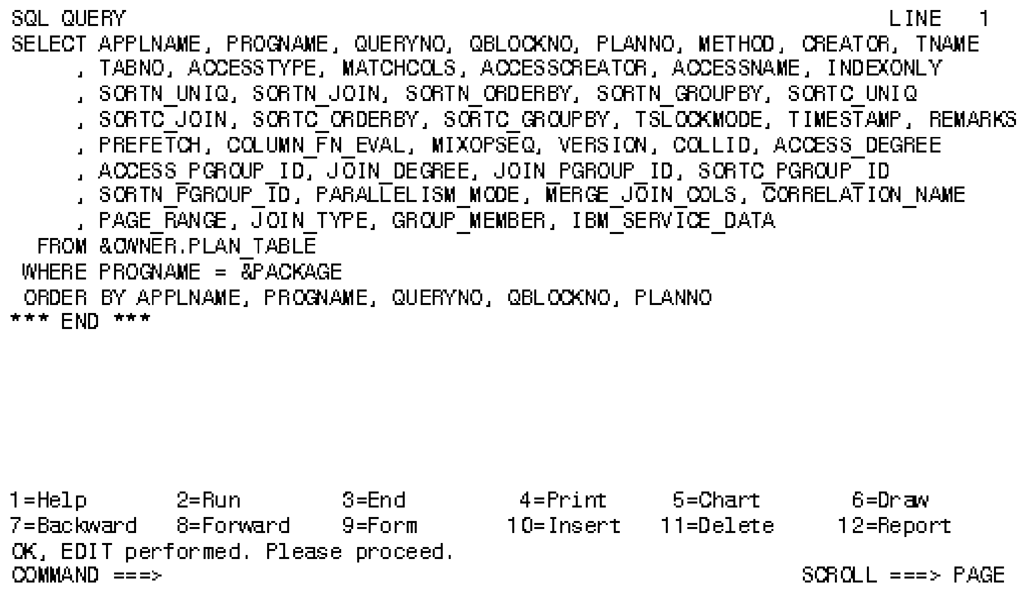
```
IBM*
 Licensed Materials - Property of IBM
 5706-254 5706-255 5648-061
 (c) Copyright IBM Corp. 1982, 1995  All Rights Reserved.
 * Trademark of International Business Machines
 ______________________________________________________________________________

  QMF HOME PANEL
  Version 3 Release 2.0                        ******     **    **      *********
                                             **     **   ***   ***     **
  Query                                     **     **   ****  ****    ********
  Management                               **     **   ** ** ** **   **
  Facility                                **   * **   **  ****  **  **
                                           ******    **   **    ** **
  Connected to                                  *
  HCFADB2T                               ______________________________________

 Type command on command line or use PF keys. For help, press PF1 or type HELP.
 ______________________________________________________________________________
 1=Help        2=List        3=End         4=Show       5=Chart        6=Query
 7=Retrieve    8=Edit Table  9=Form       10=Proc      11=Profile     12=Report
 OK, you may enter a command.
 COMMAND ===>
```

The QMF interactive environment consists of six separate panel areas (Home, Proc, Query, Form, Report, and Profile). The Q*MF Home Panel* is the first panel that is displayed when you enter a QMF session. From this panel, it is possible to enter QMF commands (such as LIST tables, or RUN QUERY). This panel can also be used to navigate to other QMF panels as well, through either DISPLAY commands or PF keys.

The *Proc Panel* provides an area to input and execute a QMF procedure (a list of QMF commands). The *Query Panel* is similar to the Proc panel in appearance only. It provides an open area to allow you to enter an SQL command. (Note the primary difference between QMF procs and queries is SQL are entered in queries and QMF commands are entered in procs). The *Form Panel* is used to format reports based on data retrieved through a query. QMF uses the *Report Panel* to merge the data retrieved by a query with the form definition in the Form panel to display a formatted output. The *Profile Panel* contains information regarding environmental settings for the QMF session. These values can be modified.

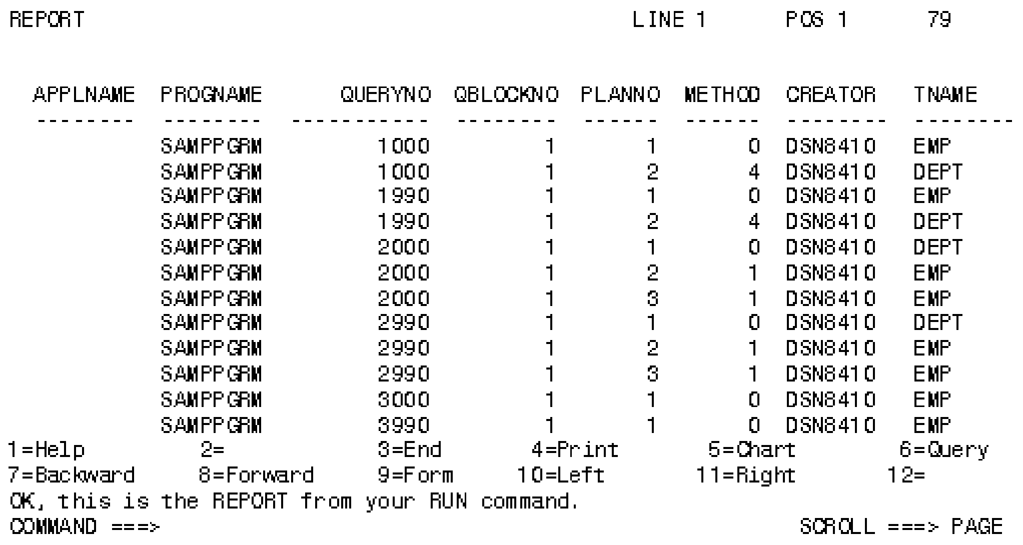
Running a Query

To run a query in QMF, you must enter the SQL statement on the QMF Query Panel. To access the query panel, enter DISPLAY QUERY on the QMF command line, or press PF6. A QMF query is an object which can be saved. It can consist of only one SQL statement therefore delimeters (like semicolons (;)) are not necessary. Once the desired SQL statement is entered in the query panel, it can be executed by pressing the PF2 key or entering the QMF RUN QUERY command. The following is an example of a QMF Query panel.



```
SQL QUERY                                                          LINE    1
SELECT APPLNAME, PROGNAME, QUERYNO, QBLOCKNO, PLANNO, METHOD, CREATOR, TNAME
     , TABNO, ACCESSTYPE, MATCHCOLS, ACCESSCREATOR, ACCESSNAME, INDEXONLY
     , SORTN_UNIQ, SORTN_JOIN, SORTN_ORDERBY, SORTN_GROUPBY, SORTC_UNIQ
     , SORTC_JOIN, SORTC_ORDERBY, SORTC_GROUPBY, TSLOCKMODE, TIMESTAMP, REMARKS
     , PREFETCH, COLUMN_FN_EVAL, MIXOPSEQ, VERSION, COLLID, ACCESS_DEGREE
     , ACCESS_PGROUP_ID, JOIN_DEGREE, JOIN_PGROUP_ID, SORTC_PGROUP_ID
     , SORTN_PGROUP_ID, PARALLELISM_MODE, MERGE_JOIN_COLS, CORRELATION_NAME
     , PAGE_RANGE, JOIN_TYPE, GROUP_MEMBER, IBM_SERVICE_DATA
  FROM &OWNER.PLAN_TABLE
 WHERE PROGNAME = &PACKAGE
 ORDER BY APPLNAME, PROGNAME, QUERYNO, QBLOCKNO, PLANNO
*** END ***




1=Help       2=Run        3=End         4=Print     5=Chart       6=Draw
7=Backward   8=Forward    9=Form       10=Insert   11=Delete     12=Report
OK, EDIT performed. Please proceed.
COMMAND ===>                                                SCROLL ===> PAGE
```

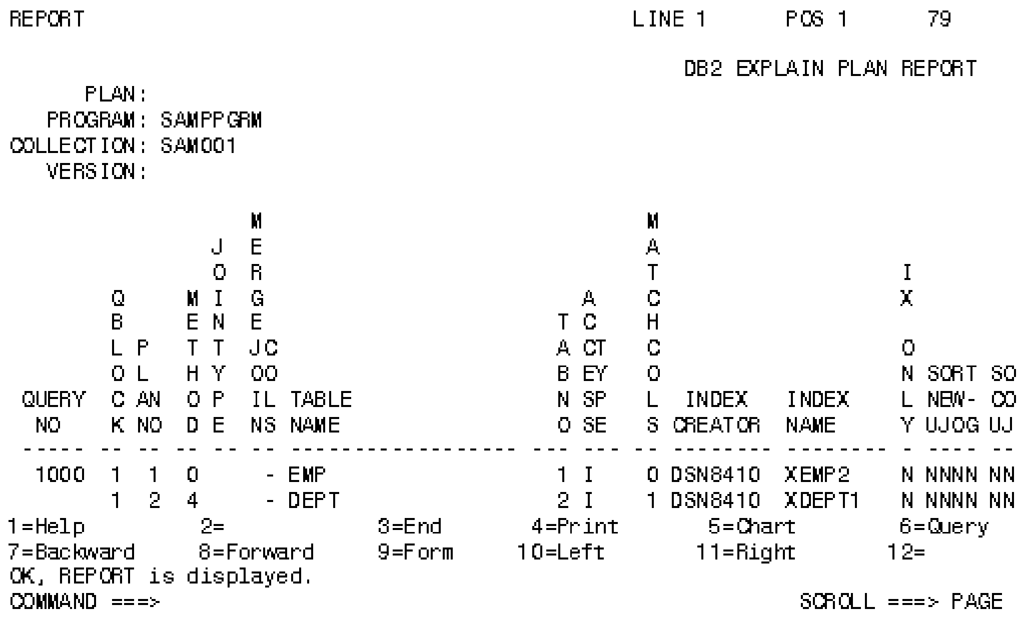
If the query runs successfully, a default report form will be generated and will be merged with the result set from the query. The merged results will be displayed on the QMF Report panel. Below is a default report format for the query above.



```
REPORT                                       LINE 1      POS 1      79

  APPLNAME  PROGNAME      QUERYNO  QBLOCKNO  PLANNO  METHOD  CREATOR   TNAME
  --------  --------  -----------  --------  ------  ------  --------  --------
            SAMPPGRM         1000         1       1       0  DSN8410   EMP
            SAMPPGRM         1000         1       2       4  DSN8410   DEPT
            SAMPPGRM         1990         1       1       0  DSN8410   EMP
            SAMPPGRM         1990         1       2       4  DSN8410   DEPT
            SAMPPGRM         2000         1       1       0  DSN8410   DEPT
            SAMPPGRM         2000         1       2       1  DSN8410   EMP
            SAMPPGRM         2000         1       3       1  DSN8410   EMP
            SAMPPGRM         2990         1       1       0  DSN8410   DEPT
            SAMPPGRM         2990         1       2       1  DSN8410   EMP
            SAMPPGRM         2990         1       3       1  DSN8410   EMP
            SAMPPGRM         3000         1       1       0  DSN8410   EMP
            SAMPPGRM         3990         1       1       0  DSN8410   EMP
1=Help        2=           3=End       4=Print       5=Chart        6=Query
7=Backward    8=Forward    9=Form     10=Left       11=Right       12=
OK, this is the REPORT from your RUN command.
COMMAND ===>                                                SCROLL ===> PAGE
```

Formatting a Report

The above report was generated by QMF by default when the query was first executed. The format of this report can be modified using QMF Form panels. The Form panel can be accessed by entering the QMF DISPLAY FORM command or pressing PF9 from the Query or Report panels. Details on modifying report formats can be found in the QMF Learner's Guide. Below is an example of a formatted report generated from the same result set noted above.



```
REPORT                                       LINE 1      POS 1      79

                                                 DB2 EXPLAIN PLAN REPORT
      PLAN:
   PROGRAM: SAMPPGRM
COLLECTION: SAM001
   VERSION:

                   M                              M
                J  E                              A
                O  R                              T                   I
        Q    M  I  G                          A   C                   X
        B    E  N  E                       T C    H
        L P  T  T  JC                      A CT   C                   O
        O L  H  Y  OO                      B EY   O                   N SORT SO
 QUERY  C AN O  P  IL TABLE                N SP   L  INDEX   INDEX    L NEW- CO
  NO    K NO D  E  NS NAME                 O SE   S CREATOR  NAME     Y UJOG UJ
 ----- -- -- -- -- -- ------------------ --- --- -- -------- -------- - ---- --
  1000  1  1 0      - EMP                  1 I    0 DSN8410  XEMP2    N NNNN NN
        1  2 4      - DEPT                 2 I    1 DSN8410  XDEPT1   N NNNN NN
1=Help        2=           3=End       4=Print       5=Chart        6=Query
7=Backward    8=Forward    9=Form     10=Left       11=Right       12=
OK, REPORT is displayed.
COMMAND ===>                                                SCROLL ===> PAGE
```

2.4.2.2 QMF Application Development (QMF Import Facility)

The QMF Import Facility is a facility available in-house which simplifies the task of developing, enhancing, testing, and migrating QMF applications. This facility allows QMF applications (procs, queries, and forms) to be stored in PDS libraries and imported to the QMF environment at execution time. By using this method, applications development teams can greatly improve their ability to manage and maintain their QMF applications. Rather than storing these applications in a DB2/QMF database for each DB2 subsystem where that application can be run, the application can be stored in one set of external libraries and referenced from any DB2 subsystem. Additionally, because these applications are stored in PDS libraries, they can be managed using the CA-Endevor configuration management tool.

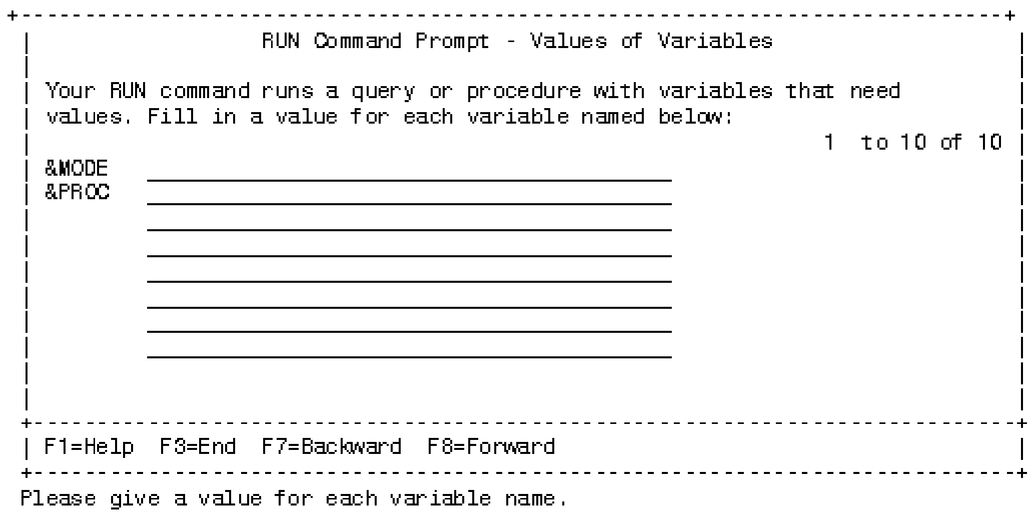
The import facility can be initiated through a batch job, clist, rexx, or from an interactive QMF session. It is initiated by executing a QMF procedure, DBA.P_QMF_INIT. The method to initiate this proc differs depending on the current execution environment. Each method is described below.

Interactive Execution

To run the initial import facility procedure from an interactive QMF session, type one of the following commands at the QMF command line:

RUN DBA.P_QMF_INIT or **QMFINIT**

Upon entering either of the above commands, you will receive the following prompt panel. Use this panel to enter all *required input parameters*.



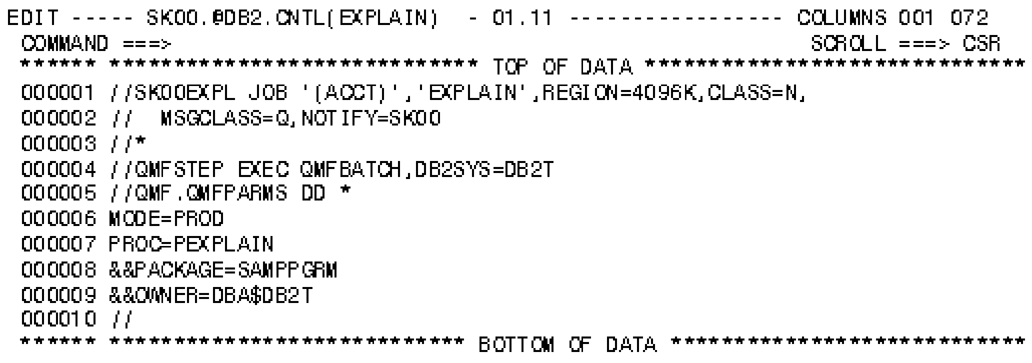
```
+--------------------------------------------------------------------------+
 |                   RUN Command Prompt - Values of Variables               |
 |                                                                          |
 | Your RUN command runs a query or procedure with variables that need      |
 | values. Fill in a value for each variable named below:                   |
 |                                                           1  to 10 of 10 |
 | &MODE     ________________________________________                       |
 | &PROC     ________________________________________                       |
 |           ________________________________________                       |
 |           ________________________________________                       |
 |           ________________________________________                       |
 |           ________________________________________                       |
 |           ________________________________________                       |
 |           ________________________________________                       |
 |                                                                          |
 |                                                                          |
 +--------------------------------------------------------------------------+
 | F1=Help  F3=End  F7=Backward  F8=Forward                                 |
 +--------------------------------------------------------------------------+
 Please give a value for each variable name.
```

After you enter the required fields and press enter. The specified QMF procedure will be imported from one of three QMF application libraries, (test, validation or production), depending on the specified MODE. The application procedure will then be executed. If the QMF import facility is unable to locate the procedure in the library specified, an error message will be displayed and the QMF session will be returned to the QMF HOME panel.

If the imported QMF application contains any substitution variables, an additional prompt panel will be displayed to accept values for each. See *Developing QMF Applications* for more details.

Batch Execution

A JCL procedure was developed to allow you to execute QMF applications in batch. The procedure, (QMFBATCH), is designed to invoke the QMF Import Facility automatically. To run the import facility in batch, set up JCL to execute QMFBATCH and provide all required input parameters . Note that the list of parameters must be provided in either a file or instream list allocated to the QMFPARMS DD. The following is an example of run JCL which will generate a formatted EXPLAIN report in batch:



```
EDIT ----- SK00.@DB2.CNTL(EXPLAIN)  - 01.11 ----------------- COLUMNS 001 072
 COMMAND ===>                                                 SCROLL ===> CSR
 ****** ***************************** TOP OF DATA ******************************
 000001 //SK00EXPL JOB '(ACCT)','EXPLAIN',REGION=4096K,CLASS=N,
 000002 //  MSGCLASS=Q,NOTIFY=SK00
 000003 //*
 000004 //QMFSTEP EXEC QMFBATCH,DB2SYS=DB2T
 000005 //QMF.QMFPARMS DD *
 000006 MODE=PROD
 000007 PROC=PEXPLAIN
 000008 &&PACKAGE=SAMPPGRM
 000009 &&OWNER=DBA$DB2T
 000010 //
 ****** **************************** BOTTOM OF DATA ****************************
```

In addition to the required parameters (MODE and PROC), values for all substitution variables present in the QMF application must also be provided in the QMFPARMS file. See *Developing QMF Applications* for more details.

Clist Execution

From within a clist the facility can be initiated by running the following statement:

```
ISPEXEC SELECT PGM(DSQQMFE) NEWAPPL(DSQE)
PARM(S=subsys,M=B,I=DBA.P_QMF_INIT)
```

Because it is necessary to provide required input parameters when invoking the import facility, a file (QMFPARMS) must be allocated to the TSO session prior to executing the above command. . This file must contain at a minimum two parameters (PROC and MODE). If the QMF application to be executed also contains substitution variables, values for each must be provided in this file as well. See *Developing QMF Applications* for more details

Rexx Execution

The QMF Import Facility can also be initiated from within a Rexx exec. QMF provides a *Callable Interface Language* that can be invoked from Rexx and other application programming languages. This feature allows applications to establish a persistent connection to QMF in order to execute QMF commands. More detailed information on the callable interface can be found in the *Developing QMF Applications* manual.

To use the QMF Import Facility from Rexx, the following steps must be performed:

1. *Establish a connection to QMF*

   From Rexx, issue a statement similar to the following to start a QMF session

   ```
   call dsqcix 'START (DSQSMODE=INTERACTIVE DSQSSUBS=DB2T DSQADPAN=0'
   ```

2. *Set required input parameters.*

   Issue QMF commands from the Rexx callable interface to establish values for PROC and MODE.

   ```
   call dsqcix 'SET GLOBAL (PROC= PEXPLAIN MODE=PROD'
   ```

3. *Set values for the any other substitution variables.*

   Issue QMF commands from the Rexx callable interface to establish values for other variables used in the QMF application

   ```
   call dsqcix 'SET GLOBAL (PACKAGE=(''SAMPPGRM'') OWNER=(DBA$DB2T)'
   ```

   * Note: When passing a quote (') to QMF from Rexx, it is necessary to pass two consecutive single quotes.

4. *Execute the QMF Import Facility.*

   Issue QMF commands from the callable interface to execute the application using the QMF Import Facility.

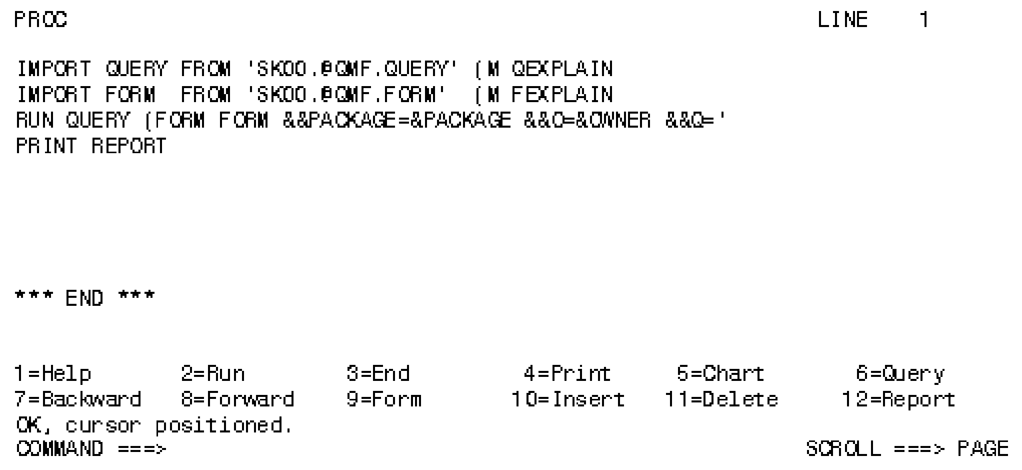
   ```
   call dsqcix 'RUN PROC DBA.P_QMF_INIT'
   ```

2.4.2.3 Developing QMF Applications

Methods used to develop QMF procedures which will utilize this facility differ slightly from those described in the *QMF Adhoc Usage* section. The primary difference is that all QMF objects developed in QMF will ultimately be stored in external PDS libraries.

A general guideline to follow when developing applications in this manner is to develop and modify all queries and forms from an interactive QMF session. Instead of issuing a QMF *SAVE* command to store the objects in the QMF environment, issue a QMF *EXPORT* command to export the query or form to a PDS. Test these objects by developing a QMF proc which will import the queries and forms from the libraries. Once you are satisfied with your results, export the proc to a PDS library and use the procedures noted above to test the application using the QMF Import Facility.

Below is a sample of a QMF proc which will import a query and a report form to generate and print a formatted EXPLAIN report.



```
PROC                                                          LINE    1

IMPORT QUERY FROM 'SKOO.@QMF.QUERY' (M QEXPLAIN
IMPORT FORM  FROM 'SKOO.@QMF.FORM'  (M FEXPLAIN
RUN QUERY (FORM FORM &&PACKAGE=&PACKAGE &&O=&OWNER &&Q='
PRINT REPORT




*** END ***


1=Help       2=Run       3=End        4=Print     5=Chart       6=Query
7=Backward   8=Forward   9=Form      10=Insert   11=Delete     12=Report
OK, cursor positioned.
COMMAND ===>                                                SCROLL ===> PAGE
```

Note that in this example the procedure will pass substitution variables to the query as it is executed. A value for each variable must be provided when this proc is invoked. The QMF Import Facility provides the ability to pass these values interactively by presenting a prompt panel, or in batch by accepting the values in a parameter file (QMFPARMS).

Required Input Parameters

The following parameters must be provided when invoking the QMF Import Facility. Depending on the execution environment (interactive or batch) you may be prompted to provide this information or will be required to provide it through a set of control cards.

Parameter	**Description**
PROC=	Name of the PDS member containing the lead QMF application procedure.
MODE=	Indicates the state of the application (test, validation, or production). This parameter determines the location of the lead QMF application procedure. Valid values are DEVL, TEST, VALD, and PROD.

When providing these parameters through a set of control cards, the following rules must be followed:

- The first two records of the control card file must contain MODE and PROC parameters (one entry per record). The order of these two records is not important.
- Each parameter name must begin in position 1 followed by an equal sign (=) and a value.
- Embedded spaces are not permitted.
- All values for substitution variables in the application procedure must be provided in the records following the MODE and PROC control cards. Values

for substitution variables must be specified in the same format required when passing variables in an interactive QMF session.

QMF Application Libraries

Mode	PDS Library
TEST	ASCM.#ENDVOUT.COMMON.QMFPROCS.DEVQPROC
VALD	ASCM.#ENDVOUT.COMMON.QMFPROCS.VALQPROC
PROD	ASCM.#ENDVOUT.COMMON.QMFPROCS.PRDQPROC

2.4.3 SAS

A SAS procedure is available at CMS which allows access to data stored in DB2. This SQL procedure (PROC SQL) includes a *"Pass-Through"* facility which passes SQL statements from a SAS application directly to the intended database management system. Any valid SQL statement can be issued to DB2 using this facility, therefore it is possible to read as well as modify data in DB2 using the methods described below.

2.4.3.1 DB2 / SAS Basics

To access DB2 from a SAS routine, use the PROC SQL procedure. This procedure supports four primary commands which allow you to send any valid SQL statement to DB2 for execution. These commands include:

CONNECT

DISCONNECT

SELECT/CONNECTION TO

EXECUTE

The CONNECT and DISCONNECT commands are used to establish and terminate conversations with DB2 from the SAS application. The SELECT command with the CONNECTION TO component is used to retrieve data from DB2. All other non-SELECT SQL statements are sent to DB2 using the EXECUTE command.

2.4.3.2 Connecting to DB2

Regardless of whether a SAS application is reading or modifying data in DB2, a CONNECT command must be issued within the PROC SQL procedure in order to establish a connection with a particular DB2 subsystem. As an example, the following statements could be used to connect to the test DB2 subsystem at CMS.

```
PROC SQL;
```

```
CONNECT TO DB2 (SSID=DB2T);
```

Once this command is executed, the application will remain connected to the DB2 subsystem until an implicit or explicit DISCONNECT command is issued (see Disconnect from DB2 for more information). Connecting to multiple DB2 subsystems within the same PROC SQL procedure is not supported.

2.4.3.3 Disconnecting from DB2

As stated above, once a connection is made to DB2 from a PROC SQL procedure, the connection will remain until an explicit DISCONNECT command is issued or until SAS implicitly terminates the connection. SAS will implicitly disconnect from DB2 when it has encountered the next PROC or DATA statement in the application flow or when the end of the application has been reached. To explicitly disconnect from DB2, issue the following SAS command:

```
DISCONNECT FROM DB2;
```

2.4.3.4 Reading DB2 Data

The CONNECTION TO component of the SELECT command is used to read DB2 data from SAS. It indicates to SAS to which database connection should the SELECT statement be directed. Since DB2 on the mainframe limits the number of connections from within a PROC SQL procedure to one, the database name or alias specified in the CONNECTION TO component must be the same as what was specified in the previous CONNECT TO statement. The following is an example of a SELECT statement that will select rows from a DB2 table.

```
SELECT * FROM CONNECTION TO DB2
     (SELECT * FROM SAMP.EMP WHERE DEPTNO = 20);
```

The first part of the above statement (SELECT * FROM CONNECTION TO DB2) is a SAS command. It will select records from the second part of the statement (the DB2 query enclosed in parenthesis). Any valid DB2 SELECT statement can be issued in this query portion of the statement. SAS limits the length of a query that it sends to DB2 to 200 characters.

When SAS passes a command like the example above to DB2, the command will be executed dynamically and the results of the query (if any) will be written to an output listing. The data is **not** sent back to SAS for further processing. In order to retrieve rows of data from DB2 into a SAS application, a SAS view must be created. The following statement illustrates this feature.

```
CREATE VIEW EMPDATA AS
```

```
SELECT * FROM CONNECTION TO DB2
        (SELECT * FROM SAMP.EMP WHERE DEPTNO = 20);
```

The above command will retrieve rows from DB2 based on the specified query (enclosed in parenthesis) and will place each row as records in the SAS view EMPDATA. EMPDATA will by default contain one field for every column selected in the DB2 query. When using the SAS CREATE VIEW statement, DB2 will not produce an output listing of the query results.

2.4.3.5 Receiving DB2 Error Messages

When an SQL command is executed in DB2 from SAS, messages from DB2 regarding the success or failure of the command are passed back to SAS. After each SQL command is executed in DB2, SAS will move return codes and possibly descriptive text to two SAS macro variables:

SQLXRC	DB2 SQLCODE
SQLXMSG	DB2 SQLCODE and corresponding text message describing the error. (Only populated when an error condition is detected).

The contents of these variables can be viewed by referencing them immediately after DB2 executes an SQL statement. For example, to print the contents of SQLXRC after issuing a SELECT statement an application could be coded as such:

```
SELECT * FROM CONNECTION TO DB2
        (SELECT * FROM SAMP.EMP WHERE DEPTNO = 20);
%PUT &SQLXRC
```

Or, if the application uses SAS views:

```
CREATE VIEW EMPDATA AS
SELECT * FROM CONNECTION TO DB2
        (SELECT * FROM SAMP.EMP WHERE DEPTNO = 20);
DISCONNECT FROM DB2;
PROC PRINT DATA=EMPDATA;
%PUT &SQLXRC
```

Note that in the latter example, the reference to &SQLXRC is made immediately after the view (EMPDATA) is accessed. This is because the DB2 query that corresponds to this view is not executed until SAS makes its first reference to the SAS view.

2.4.3.6 Modifying DB2 Data

The EXECUTE command in SAS allows an application to execute non-SELECT statements against data in DB2. The format of an EXECUTE statement is as follows:

EXECUTE (*sql statement*) BY DB2;

To use this command, the application must first connect to the target DB2 subsystem. As with the SELECT statement, it is possible to view the result of the executed SQL statement by inspecting the macro variables, SQLXRC and SQLXMSG.

2.4.4 Quickstart / MVS

Quickstart/MVS is a tool from BMC Software that allows batch application programs to synchronize resources (sequential files, DB2 tables, etc.) it utilizes while processing. This tool provides uniform processing logic to allow batch application programs to be restarted in the event of an application failure. The tool acts much like a DBMS in that checkpoints can be issued during program execution to record the completion of a unit of work. If the application ends abnormally, processing can resume at the last previously recorded checkpoint.

Many approaches can be used to incorporate the Quickstart/MVS tool in an application. The following documents procedures which can be used to include this tool with batch DB2 application programs. Since the primary focus of this information is on synchronizing DB2 resources, please refer to the Quickstart*/MVS User Guide* for details on the product use with resources outside of DB2

2.4.4.1 Overview

Quickstart/MVS, when used in conjunction with DB2, performs the majority of the processing logic necessary to achieve commit/restart capability within a batch application program. When initiated, via a checkpoint routine, it will record the status of an executing application program, store the contents of specified program variables, note the position into sequential files, and issue a DB2 COMMIT. If the application program ends abnormally, the status of the program at the last checkpoint will be retained. Upon the restart of the application program, Quickstart will recall the last checkpoint information, restore the state of the saved program variables, and reposition all sequential files, enabling the program to resume processing.

2.4.4.2 Program Enhancements

Quickstart/MVS provides two different methods to use the product with application programs, *Transparent Mode* and *API Mode*. Since API Mode provides the highest amount of flexibility, this mode should be used when developing applications at CMS.

The following program enhancements must be made in order to use Quickstart/MVS in API Mode and to ensure the application is in fact restartable. See the *Examples* section of this document for additional information.

1. Identify files which must be repositioned in the event of a restart condition.
2. Convert repositionable files to use Quickstart/MVS COBOL FD Interception.
3. Copy Quickstart/MVS copybooks, QSWSLIT, QSWSBEG, and QSWSEND into the application program.
4. Identify application program Working Storage areas which would need to be restored in the event of a restart condition.
5. Code performable paragraph in the Procedure Division of the program to CALL the Quickstart/MVS checkpoint module, CKPTRTN .
6. Code initialization logic to perform the checkpoint paragraph (step 5) to initialize Quickstart/MVS and to check for a restart condition.
7. Code "wrap-up" logic to perform the checkpoint paragraph (step 5) to perform a "normal" termination of Quickstart/MVS.
8. Design application logic to incorporate Logical Unit of Work (LUW) techniques. Incorporate logic that will periodically perform the checkpoint paragraph (step 5) to issue a checkpoint and record program status information.
9. When appropriate, code cursor definitions such that the cursor position will not be lost during checkpoint processing and can be repositioned during program restart processing.
10. Add logic to DISPLAY checkpoint and restart information. This information would be extremely useful to determine the progress and successful execution of your application. Additionally, this information may be presented during reviews of the application to document abend/restart capability.

2.4.4.3 Program Preparation

The program preparation process for DB2 application programs does not differ when the program is used in conjunction with Quickstart/MVS. There is a minor difference to the way the batch application plan is bound (plan must include the collection id (QUICKSTART.*) for Quickstart/MVS packages), however this will be handled by the central DBA.

2.4.4.4 JCL Enhancements

The following JCL modifications must be made to job streams which execute application programs using Quickstart/MVS.

1. Add the Quickstart/MVS executable load library to the appropriate steplib or joblib. This library name is CMS1.@QSTART.P1.LOADLIB. (Note, this library name may differ depending on the corresponding DB2 subsystem. Check with the central DBA to determine the correct name for your executing environment.)

2. Add a *PARMLIB* DD card to enable Quickstart to dynamically allocate a Checkpoint Dataset. Quickstart uses this file to store Working Storage and file positioning information. The dataset name specified on the DD card is used by Quickstart to determine how to name the checkpoint dataset it creates dynamically. The name can contain any high-level qualifiers with the last qualifier equal to the jobname. The DD card should be coded as follows:

   //PARMLIB DD
   DSN=*hlq1.hlq2.jobname*,SPACE=(trK,0),UNIT=*unit*

3. Adjust the disposition of any output sequential datasets that would need to be repositioned in the event of a restart scenario. Disposition of these files should be set to MOD,CATLG,CATLG. Upon the initial execution of your program, Quickstart will initialize these datasets as if the disposition were NEW. Subsequent restarts of the job will force Quickstart to append the datasets, using the MOD disposition.

   Note: For restart purposes, report files can also be written to output datasets using this method instead of writing them to SYSOUT.

2.4.4.5 Examples

Program Enhancements

The following are excerpts from a COBOL/DB2 application program which uses Quickstart/MVS in API Mode. Note that all sample COBOL code written in **bold** print is required.

1. Identify files which must be repositioned in the event of a restart condition.

2. Convert repositionable files to use Quickstart/MVS COBOL FD Interception. *Note: This step is only required for non-DBMS data files. Repositioning of DB2 data will be handled automatically*.

   COPY QSOPENI REPLACING MYFD BY CARDIN.

   COPY QSOPNADV REPLACING MYFD BY REPOUT.

   ...

   COPY QSCLOSEI REPLACING MYFD BY CARDIN.

   COPY QSCLSADV REPLACING MYFD BY REPOUT.

3. Copy Quickstart/MVS copybooks, QSWSLIT, QSWSBEG, and QSWSEND into the application program.

   - *QSWSLIT* contains a list of program literals used to invoke the Quickstart

APSIs.

- *QSWSBEG* marks the beginning of the Working Storage Checkpoint Save Area -- the area that will be saved by Quickstart at each invocation of the CKPTRTN module. These areas will be restored by Quickstart at program restart.
- *QSWSEND* marks the end of the Working Storage Checkpoint Save Area.

4. Identify application program Working Storage areas which would need to be restored in the event of a restart condition. Position each of these areas in Working Storage between copybooks, *QSWSBEG* and *QSWSEND*. All data elements coded between these copybooks will be saved in the Checkpoint Dataset.

```
01  QS-PROGRAM-LITERALS.
    02 LUW-COUNTER              PIC S9(4) COMP VALUE +0.
    COPY QSWSLIT.
    COPY QSWSBEG.
    02 REC-PROCESSED-COUNTER    PIC S9(4) COMP VALUE +0.
    02 RECS-LAST-PROCESSED      PIC S9(4) COMP.
    02 WS-LAST-ORG-EIN          PIC X(9).
    02 WS-LAST-ORG-NAME         PIC X(35).
    COPY QSWSEND.
```

5. Code performable paragraph in the Procedure Division of the program to CALL the Quickstart/MVS checkpoint module, CKPTRTN.

```
CKPT-RTN.
*
** THE FOLLOWING CALL TO CKPTRTN HANDLES ALL CHECKPOINTING
** NEEDS, INCLUDING SAVING SEQUENTIAL FILES POSITIONS, SAVING
** WORKING STORAGE, AND ISSUING A DB2 COMMIT.
*
DISPLAY 'BEGIN CKPT-RTN. RECORD COUNT = ' REC-PROCESSED-COUNTER.
CALL 'CKPTRTN' USING CKPT-SAVE-AREA, CKPT-AREA-END.
DISPLAY 'END CKPT-RTN. RECORD COUNT = ' REC-PROCESSED-COUNTER.

CKPT-RTN-EXIT.
EXIT.
```

6. Code initialization logic to perform the checkpoint paragraph (step 5) to initialize Quickstart/MVS and to check for a restart condition.

```
MOVE 'pgmname' TO CKPT-PGM-NAME.
PERFORM CKPT-RTN THRU CKPT-RTN-EXIT.
```

7. Code "wrap-up" logic to perform the checkpoint paragraph (step 5) to perform a "normal" termination of Quickstart/MVS.

```
MOVE 'E' TO CKPT-REQUEST-TYPE.
PERFORM CKPT-RTN THRU CKPT-RTN-EXIT.
```

8. Design application logic to incorporate Logical Unit of Work (LUW) techniques. Include logic that will periodically perform the checkpoint paragraph (step 5) to issue a checkpoint and record program status information.
Note PARM-IN-COMMIT-FREQ is an input parameter read in by the program indicating how often a commit (Quickstart checkpoint) should be performed. This method is recommended to allow the commit frequency to be modified at application run time.

```
ADD 1 TO REC-PROCESSED-COUNTER, LUW-COUNTER
IF LUW-COUNTER < PARM-IN-COMMIT-FREQ

CONTINUE

ELSE

MOVE REC-PROCESSED-COUNTER

TO RECS-LAST-PROCESSED

MOVE 'F' TO CKPT-REQUEST-TYPE

MOVE ZEROS TO LUW-COUNTER

PERFORM CKPT-RTN THRU CKPT-RTN-EXIT

END-IF
```

9. When appropriate, code cursor definitions such that the cursor position will not be lost during checkpoint processing and can be repositioned during program restart processing.
When coding a cursor that must be repositioned, care must be given to the sequence of the data being retrieved. Code the ORDER BY clause in such a way that will make it easy to determine the last row processed. The WHERE clause can then include additional search criteria to retrieve only those rows that have not yet been processed. The following example illustrates this technique. Note, the WITH HOLD option must be specified in order to retain the current cursor position each time a checkpoint (DB2 COMMIT) is processed.

```
DECLARE ORG_LIST CURSOR WITH HOLD FOR
SELECT ORG_NAME, ORG_EIN, ORG_PHN_NUM

FROM AAA_ORGANIZATION

WHERE ORG_NAME LIKE :WS-ORG-NAME

AND ORG_NAME >= :WS-LAST-ORG-NAME

AND ORG_EIN > :WS-LAST-ORG-EIN

ORDER BY ORG-NAME,ORG-EIN
```

10. Add logic to DISPLAY checkpoint and restart information. This information would be extremely useful to determine the progress and successful execution of your application. Additionally, this information may be presented during reviews of the application to document abend/restart capability.

```
IF PROGRAM-IS-RESTARTING
DISPLAY 'RESTART IN PROGRESS AT RECORD ',

RECS-LAST-PROCESSED

ELSE

DISPLAY 'BEGIN PROCESSING ',

'RECORD COUNTER=' REC-PROCESSED-COUNTER

END-IF
```

Naming Standards

3.1 Conventions for Objects and Datasets

This section discusses the standard naming conventions for DB2 objects and datasets. These conventions were designed to meet the following objectives:

· Guarantee uniqueness of DB2 object names within a DB2 subsystem

· Provide a uniform naming structure for DB2 objects of similar types

· Simplify physical database design decisions regarding naming strategies

· Provide ability to visually group DB2 Objects by the application and/or subject area for which the objects were designed to support

The following naming standards apply to any DB2 object (database, tablespace, table, view, package, etc.) used to hold or maintain user data, as well as external user objects, (partitioned datasets, production job names, etc.) that will be used in conjunction with DB2 as part of standard application development and system operation procedures.

3.1.1 Standard Naming Format for DB2 Objects

All DB2 objects will be named according to the standard formats listed in the following table. All object names must begin with an alphabetic character and cannot begin with DSN, SQL, or DSQ.

DB2 Objects / **Object Type**	**Required Attributes**	**CMS Standard Format**	**Example**
Alias	• 18 character (maximum); • alias name will always match the name of the table to which the alias pertains	creator.aliasname	DBA$DB2T. NPS_PROV_INFO
column	• 18 character (maximum); • unique within the corresponding table • name should be derived from the business name identified during the business/data analysis process; • name must include acceptable class and modifying words as defined by the data	bb...bb	PROV_ID

	administration group.		
Collection	• 6 character name • The first three-characters (AAA) is the Application Identifier • The last three-positions is always '001' • Plans are always created by the Central DBA	AAAttt	NPS001
Database	• 8 characters name (maximum); • First three characters represent *Subject Area* under which the data contained within the database is categorized • the next three characters indicate the application responsible for the data; this abbreviation must be approved by Central DBA and APCSS • the last two positions are sequential numbers used to uniquely identify the database within subject and application	sssAAAnn	PRVNPS01

	• All Databases are defined by the Central DBA		
DBRM	• 8 character name (maximum), (three-character Application Identifier & five-char description); • DBRM name must be identical to the source module name of the corresponding application program.	AAAbbbbb	NPSPROG1
DCLGEN	• 6-7 character library member name • for tables, the DCLGEN member name will correspond to the name of the corresponding tablespace. • for approved views, the DCLGEN member name should begin with a one-position alphabetic character ('V' for 'view'), a three-character Application Identifier, and a three-position sequential number uniquely identifying the view member	AAAttt - table VAAAttt - views	NPS001 VNPS001

	name		
Foreign Key	• 8 character name; • three-character Application Identifier, a three-position sequential number indicating the corresponding tablespace number, a one-position fixed character ("F"), and a one-position sequential number indicating the specific foreign key for the corresponding table/tablespace ; • must be unique within the corresponding database definition.	AAAtttFn	NPS001F1
Index	• 8 character name; • three-character Application Identifier, a three-position sequential number indicating the corresponding tablespace number, and a two-position sequential number indicating the specific index for the corresponding	AAAtttnn	NPS00101

	table/tablespace ; • must be unique within database		
Package	• 8 character name (maximum), (three-character Application Identifier & five-char description); package name must be identical to the source module name of the corresponding application program.	AAAbbbbb	NPSPROG1
Plan	• CMS utilizes two plans for each application one for online programs and one for Batch programs • 7 character name • The first three-characters (AAA) is the Application Identifier • The fourth is either a 'C' for CICS or a 'B' for non-CICS • The last three are always '001' • Plans are always created by the Central DBA	AAAcnnn	MSIB001
Program	• 8 character name (maximum),	AAAbbbbb	NPSPROG1

	(three-character Application Identifier & five-char desc); • use existing standards (refer to *CMS Data Center Users' Guide*		
schema	• 6 character name, the name is the same as the collection used for the application	See collection	See collection
Storage Group	• 8 character name. • First character must be alphabetic, the following characters can be either alphabetic or numeric; • Name must be unique within subsystem • Storage groups are defined by the Central DBA staff	bbbbbbbb	STOGRPT1
Stored Procedure	• 8 character name • Three-character application identifier, a two position fixed character "SP", a three position alphanumeric program id.	AAASPbbb	MBDSP001
Table	• 18 character name (maximum); • three-character	AAA_bb..bb	NPS_PROV_INFO

	Application Identifier followed by an underscore (_) and up to 14 alphanumeric characters describing the contents of the table. • Table names must be created in the same manner as column names and must follow the same rules. Refer to the data administration standards for more information. • Table description should include prime word as described in the *CMS Information Systems Development Guide*. Note: table name is qualified by an 8 character object creator Id.		
Table Check Constraint	• 18 character name; • three-character Application Identifier, a three-position sequential number indicating the corresponding tablespace number, an	AAAttt_bbbbbbbbbbb	NPS001_CLASS_CODE

	underscore (_), and up to 11 alphanumeric characters describing the constraint. • must be unique within corresponding table definition.		
Table Creator Id	• 8 character name • First four characters are “DBA$” • Last three characters is the DB2 subsystem Id	DBA$AAA	DBA$MED
Tablespace	• 6 character name; • three-character Application Identifier and a three-position sequential number indicating the specific tablespace number; • must be unique within database	AAAttt	NPS001
Trigger	• 7 character name • 1-3 (AAA) is the Application Id • 4 (t) describes the trigger type (B-efore or A-fter) • 5-7 (nnn) Tablespace number • 8 (x) Sequence number - 0-9 A-	AAAtnnnx	DSYB0210 for a BEFORE trigger on tablespace DSY021

	Z		
UDF and Distinct Types	• UDF's and distinct types are not currently supported at CMS, contact the Central DBA staff for guidance		
View	• 18 character name (maximum); three-character Application Identifier followed by an underscore (_) and up to 14 alphanumeric characters describing the contents of the view • Use of views must be approved by the Central DBA group Note: view name is qualified by an 8 character object creator Id which, on the test DB2 subsystem, is determined by the secondary authorization Id of the Project DBA creating the view. In production, the object creator Id for all views is DBA$DB2P.	AAA_bb..bb	NPS_ACTV_PROV

3.1.2 Standard Application Identifiers

Standard names for most DB2 objects and datasets include a three-character code that indicates to which application the object belongs. Exceptions to this naming convention are subsystem IDs, storage groups, and columns.

The Application Identifier identifies a CMS-specific system or application. The APCSS, Central DBA Group and application development teams jointly determine and assign the three-character code. Contact APCSS for a list of assigned application identifiers.

3.2 DB2 Subsystem Names

The Subsystem ID is a unique four-character name used to identify a DB2 address space to the MVS operating system. Within each DB2 subsystem, there is a directory and catalog containing all of the DB2 objects (i.e., tablespaces, tables, indexes, application plans, etc.) defined within.

The distinct DB2 subsystem environments, their standard DB2 names, usage and availability are shown in the table below. The naming standard for DB2 subsystem environments is a four-digit code where the first three digits are 'DB2' and the last digit identifies the subsystem as follows.

Standard Name	**Subsystem Environment**			**Object Names**
	Type	**Usage**	**Available**	**ALIAS/Creator Name**
DB1T DB2T DB3T	Test & Development	Primarily used by DBAs and applications programmers to design and build relational databases and the applications that access those databases.	daily	DBA$DB1T DBA$DB2T DBA$DB3T
DB2I	Integration & Stress Testing	For final pre-validation testing and performance measurement.	daily	DBA$DB2I
DB1V DB2V DB3V	Validation	DBA, application programmers, end-users access DB2 relational databases to verify all components of an application prior to production implementation.	daily	DBA$DB1V DBA$DB2V DBA$DB3V
DB1P	Production	End-users access operational DB2 relational	daily	DBA$DB1P

DB2P DB3P		databases.		DBA$DB2P DBA$DB3P
DB2W	Data Warehouse	End-users access decision support/warehoused DB2 relational databases	daily	DBA$DB2W

3.3 Production Library Names

The following is a list of standard library names to be used by applications developers and local and central DBAs.

Library Name	Description
PROD.DBRM.LIB	This library contains the database request modules produced by the DB2 precompiler, which are input to the BIND process.
PROD.COPY.LIB	This library contains the output from DCLGENs. The member name should be the name of the table or view. Each member contains the DB2 object columns and its COBOL copybook representation. This is a PANVALET structure.
PROD.JCL.LIB	This library stores the batch JCL used to submit standard Production DB2 utility jobs.
PROD.PROC.LIB	This library holds the cataloged JCL procedures (i.e., application) jobstreams for all DB2 applications.

3.4 Test Library Names

The following is a list of standard library names used by Endevor for development.

Library Name	Description
ASCM.#ENDVOUT.xxx.yyyyyy.DEVDBRM Where xxx is the application ID and yyyyyy is the Endevor subsystem	This library contains the database request modules produced by the DB2 precompiler, which become input to the BIND process.
ASCM.#ENDVOUT.xxx.COPYLIB.DEVCOPY	This library contains the output from DCLGENs. The member name should be the name of the table or view. Each member contains the DB2 object columns and its COBOL copybook representation. This is a PANVALET structure.

3.5 Image Copy Dataset Names

The naming standard is the same for both system and user image copy datasets as follows:

yyyy.LLL.dddddddd.tttttttt.Annn.GxxxxVxx, where:

yyyy	DB2 Subsystem ID (e.g., DB2T, DB2P, etc)
LLL	designates location of imagecopy dataset (LOC = local, DRV = disaster recovery, or OFS = offsite archive vault 2) The storage medium (Tape, DASD) will be directed by media management based on a DSN pattern.
dddddddd	eight-character database name
tttttttt	eight-character tablespace name
Annn	partition number (A001 for non-partitioned tablespaces)
I	type of copy (F = full, I = Incremental) if both types are used. No type implies full copy.
GxxxxVxx	generation dataset group number

Local Copy Example: DB2T.LOC. PRVNPS01.NPS001.A001.G0708V00

- At least two production copies will be made, one local and one to an offsite vault.
- Test only requires one copy locally. When the test file is not easily rebuilt from production, an offsite copy of test data should be sent to the archive vault. Test copies should not be in the disaster recovery vault, unless permission is given by the central DBA and media management group.
- For full image copies, at least 3 generations should be kept.
- For incremental copies, all generations after the previous full copy should be kept.
- The same number of generations is not required for onsite and offsite copies. Disaster recovery should be limited to 3 full copy generations even when additional copies are kept locally.
- Production files, even if read only, require an offsite copy at least once a year, unless provision for rebuilding the file from source data is provided to APCSS.
- When hotsite recovery is required, image copies must be made at least every six months. This insures that at least two backups exist at the hotsite in the event that a tape error prevents the first tape from being read.
- Image copy files may only be kept for three years.
- Catalog control, EXPDT=99000, will be used to control expiration dates.

3.6 Utility Job Names

A utility ID is a name used to uniquely identify a utility in the DB2 subsystem, only one instance of a unique utility ID may be active within DB2 at any given time. The standard for naming utilities will be to use the jobname. For multi-step and/or multi-utility jobs, the first seven characters of the jobname and 1, incremented by 1, for each successive step will be used.

Utility ID Format: Example: Jobname MSI#01IC (image copy with three steps running one utility per step)

Utilids: MSI#01I1, MSI#01I2, MSI#01I3

DB2 Utility Type Codes

Code	Description
IC	Image Copy
RG	Reorg
RC	Recover
RI	Recover Index
RS	Runstats
QU	Quiesce
MD	Modify
RB	Rebuild Index

3.7 DB2 / ORACLE Naming Issues

This section is intended for applications that implement common data in both DB2 and ORACLE. Objects that are replicated in both databases must be named the same. Please refer to the EDG/DSS/DA naming standards.

Security

4.1 DB2 Security Administration

STANDARD

DB2 Security at CMS is administered using RACF groups. Individual userids are associated with one or more RACF Groups, each having a different set of authorities and privileges in DB2. **All central DBAs, local DBAs, application developers,**

and application users must be assigned to and use RACF Groups within DB2. DB2 privileges and authorities are not granted to individual userids.

The following sections describe the DB2 security procedures for each of the DB2 processing environments.

4.1.1 Development

In the DB2 Development environment, security will be administered at four levels (see *DBA Roles and Responsibilities* for more details). They are:

- Central DBA
- Local DBA
- Application Developer
- Application User

Upon the initiation of a DB2 application development project, the *central DBA* will perform several steps to prepare the development environment for the Application Development team. These activities include:

- create RACF groups specifically for the new application;
- create a DB2 database which will be used to group all DB2 objects created for the application;
- grant preliminary privileges to the project team;
- create online and/or batch application plans;

The *local DBA* is primarily responsible for maintaining database objects for the application. Once the initial database environment is turned over to the local DBA by the central DBA, object creation and maintenance can begin. In addition to creating objects, the local DBA must perform tasks to ensure that the development and user groups have appropriate access to this new environment. These tasks include but are not limited to:

- connect individual userids with the appropriate RACF Groups;

Each of the tasks noted above are described in the sections that follow.

4.1.1.1 RACF Groups

The central DBA will create RACF Groups which will be used to administer security on all objects created for the application. Central DBA will associate these RACF groups (listed below) along with the common RACF Group for the development environment, to the userid of the local DBA assigned to the application.

Development RACF Groups

DBA$DB2T	*Common development RACF Group*. This group is used to create DB2 ALIAS' for all tables and views created in the development environment

DBA$*xxx*	RACF Group to be used by the *local DBA* to manage DB2 objects.
DEV$*xxx*	RACF Group to be used by the *Application Developer* to manipulate DB2 data, bind application packages, execute specific database utilities.
USR$*xxx*	RACF Group to be used by the *Application User* in the development environment. This RACF Group should facilitate user testing activities.

Where xxx is the assigned application identifier.

The central DBA will also grant the local DBA connect authority within RACF so that the local DBA may add or remove userids to/from the DEV$*xxx* and USR$*xxx* RACF Groups. The central DBA will add or remove individual RACF ids to/from the DBA$*xxx* and DBA$DB2T RACF Group as requested by the designated local DBA and the Application's management.

4.1.1.2 Application Database

The central DBA will create at least one database that will be used to manage the DB2 objects (tables, tablespaces, indexes, etc.) for the application. The local DBA will be responsible for the creation and maintenance of all DB2 objects within this database. The local DBA must use authorities associated with the DBA$*xxx* RACF group to accomplish this task.

To create or maintain an application database object using DDL (data definition language), the local DBA must first connect to the local DBA RACF group (DBA$*xxx*) using the SET CURRENT SQLID command. This will ensure that the local DBA has the appropriate authority to create or manage the object and that all objects created are owned by the local DBA RACF group. For each database table or view created, the local DBA must also create a DB2 alias using the common development RACF group (DBA$DB2T). This is necessary to facilitate the required use of unqualified table/view names from application programs, permit a more generic and flexible BIND process, and allow for a seamless migration of database objects from the development environment to subsequent DB2 subsystems.

4.1.1.3 Preliminary Privileges (central DBA)

As part of the initial steps in setting up a DB2 application database in the Development environment, the central DBA will grant the following privileges.

Local DBA Privileges

After creating the application database, the central DBA will grant DBADM authority on the new database to the local DBA RACF group (DBA$*xxx*). With this authority, the local DBA will have the ability to create and maintain all database objects needed for the application within this new database. The local DBA will not be given authority to create a database.

In addition to granting DBADM authority, the central DBA will also grant to the local DBA RACF Group the following database level privileges.

Database Privileges granted

DISPLAYDB	issue the DB2 -DISPLAY DATABASE command
IMAGCOPY	execute the COPY utility to create a backup of a tablespace
LOAD	execute the LOAD utility
STATS	execute the RUNSTATS utility to update catalog statistics
STARTDB	issue the DB2 -START DATABASE command
STOPDB	issue the DB2 -STOP DATABASE command

The above privileges may be augmented or removed by the central DBA as experience dictates.

Application Developer Privileges

Central DBA will grant DB2 system-level privileges, BINDADD and CREATE IN COLLECTION '*xxx*nnn', to the Application's Developer RACF Group (DEV$*xxx*). All packages created by the application development team must be created using this assigned collection id.

A single database (DEVPLN01) and tablespace (PLN001) exists for the purpose of storing plan tables for the application development environment. As an additional step, the central DBA will grant CREATE table privilege in this database to the Application Development RACF group. This will permit the developer to create a plan table for their application.

4.1.1.4 Application Plans

The central DBA will create application plans in DB2 which can be used to associate application programs (DB2 packages). Plans will be created for the online and batch environments and will be named *xxx*C00n and *xxx*B00n respectively (where xxx is the assigned application identifier and n is a number). Plans will contain a generic package list as part of their definitions which will include all packages created under the collection id(*xxx*00n) assigned for the application.

The following authorities are set up for each plan:

Online	EXECUTE ON PLAN *xxx*C00n TO USR$*xxx*
Batch	EXECUTE ON PLAN *xxx*B00n TO DEV$*xxx*

4.1.1.5 Application Packages

Application Developers are responsible for binding all DB2 application packages in the development environment. In order to do this, certain DB2 privileges are required. The DB2 system-level privileges of BINDADD and CREATE IN COLLECTION '*xxx*001', (where xxx is the assigned *application identifier*), will be granted to the Application's Developer RACF Group (DEV$*xxx*). All packages created by the application development team must be created using this assigned collection id.

Binding an application package in the DB2 development environment is accomplished automatically whenever a DB2 application program successfully compiles using CA-Endevor. Endevor processors are in place to associate the application currently being prepared with the appropriate DB2 collection id (*xxx*001).

When a BIND is executed using Endevor, the developer's RACF group (DEV$*xxx*) will be used to verify authorizations within DB2.

4.1.1.6 Plan Table (Application Developers)

Tablespaces (DEVPLN01.PLN001) and (VALPLN01.PLN001) exist for the purpose of storing Explain Tables in all of the Development (DB*T) and Validation (DB*V) Environments.

The Central DBA is responsible for creating the four(4) Tables required in each of the Environments referenced by the Endevor path(s) defined for the Application. This should be done when RACF authority is requested for a NEW application, or as soon thereafter as practical.

The JCL to create the required Tables, as well as instructions for substitutions and execution may be found in TEST.JCL.LIB (EXPLTABL).

The Template used to support the Table creation process will be maintained at the current DB2 release level by the Central DBA Group. All tables created at prior release levels are upward compatible and no intervention is required for past releases at BIND time.

Note: Each application development group must have Tables PLAN_TABLE, PLAN_HST, DSN_STATEMNT_TABLE, and DSN_STATEMNT_HST defined in the environment referenced by the current path for the CA-Endevor stage in order to successfully BIND DB2 Packages using CA-Endevor program preparation procedures.

DSN_FUNCTION_TABLE and DSN_FUNCTION_HST may be created individually for applications containing user defined functions if desired, but mechanized maintenance of these tables is not currently supported by Endevor or the Central DBA Staff.

4.1.1.7 Catalog Access

All DB2 users in the test DB2 subsystems will have SELECT authority on most DB2 catalog tables.

4.1.1.8 Central DBA Policies

The central DBA Group maintains overall control of all DB2 subsystems. While there is no predetermined limit to the number of databases a local DBA may control within an application area, central DBA will require justification/explanation for the need.

It is the local DBA's responsibility to create and monitor the number of objects housed within a given database. In the event database definitions (DBDs) or any other resource begins to impact performance of the EDM pool and/or other subsystem activity, central DBA will initiate necessary corrective actions, to include a review of subsystem activity, any necessary resource maintenance, and investigation of all anomalies such as unusually large DBDs. These activities can and should be initiated by any local DBA that feels that their development efforts are being impeded.

4.1.2 Validation

In the DB2 Validation environment, security will be administered at three levels (see *DBA Roles and Responsibilities* for more details). They are:

- Central DBA
- Local DBA
- Application User

Migration of DB2 database objects from the DB2 development environment to the validation environment is the responsibility of the *central DBA*. The central DBA will perform several steps to prepare the application database in the DB2 validation subsystem. These activities include:

- migrate all DB2 database objects, (database, tablespaces, tables, indexes, etc) from the development environment;
- grant privileges to the local DBA and when necessary;
- create online and/or batch application plans.

In the validation environment, the *local DBA* is primarily responsible for operational issues regarding the application database objects. The local DBA must execute all necessary database utilities (load, copy, reorg, etc) responding to problems when required.

For most host-based applications (COBOL) developed at CMS, *Application Users* will not need to have specific privileges permitting them to access DB2 tables directly. For the most part, this group will be granted permission to execute DB2 applications only. For Application Users who must access DB2 using Dynamic SQL, (QMF, SAS, Visual Basic), these privileges may be added as needed.

Each of the tasks noted above are described in the sections that follow.

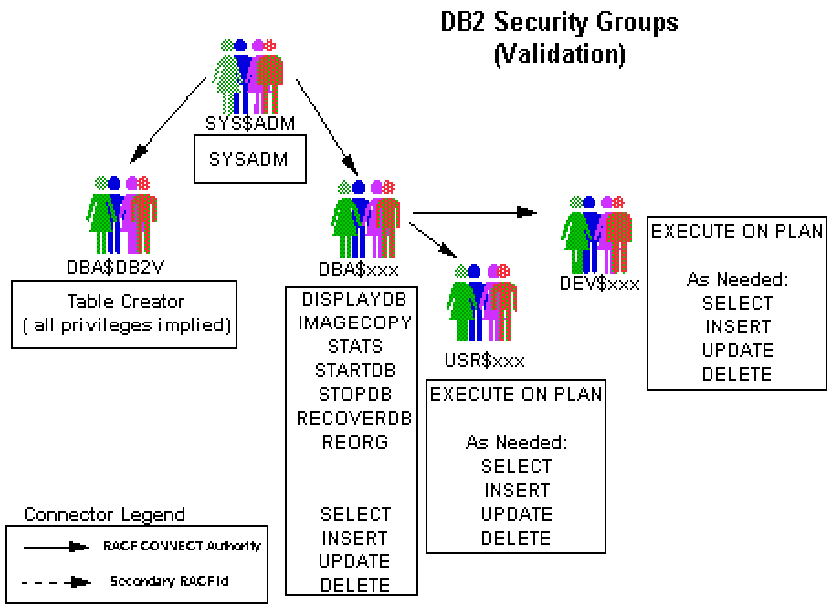
4.1.2.1 RACF Groups

In the Validation environment, the central DBA will use the Common Validation RACF Group to create and maintain all application database objects. The RACF groups listed below for the *local DBA* and the *Application User* will have limited authority in the Validation environment. See RACF Groups in the *Security (Development)* section of this document for details on how these RACF groups are assigned.

Validation RACF Groups

DBA$DB2V	*Common Validation RACF Group*. This group is used to create all DB2 objects in the validation environment.
DBA$*xxx*	RACF Group to be used by the *Project DBA* to execute utilities and monitor DB2 objects.
DEV$*xxx*	RACF Group to be used by the *Application Developer* to manipulate DB2 data, execute specific database utilities, and test application programs as required.
USR$*xxx*	RACF Group to be used by the *Application User* in the validation environment. This RACF Group should facilitate user testing activities.

where xxx is the assigned application identifier



4.1.2.2 Application Database

The central DBA will create all database objects in the DB2 Validation environment for the migrating application. All objects created in this environment will have an associated creator id of DBA$DBxV (Where x=subsystem number). The local DBA will not have the authority to create, alter, or drop any database objects in the Validation environment.

4.1.2.3 Preliminary Privileges (Central DBA)

As part of the initial steps in setting up a DB2 application database in the Validation environment, the central DBA will grant the following privileges.

Local DBA Privileges

After creating the application database objects in the Validation environment, the central DBA will grant the local DBA RACF Group the following database level privileges.

DISPLAYDB	issue the DB2 -DISPLAY DATABASE command
IMAGCOPY	execute the COPY utility to create a backup of a tablespace
LOAD	execute the LOAD utility to bulk load data
STATS	execute the RUNSTATS utility to update catalog statistics
STARTDB	issue the DB2 -START DATABASE command
STOPDB	issue the DB2 -STOP DATABASE command
RECOVER	execute the RECOVER utility to restore application data
REORG	execute the REORG utility to optimize storage utilization

In addition to the privileges noted above, the central DBA will grant the local DBA RACF group INSERT, UPDATE, DELETE, and SELECT authority on each application table

Note: The above privileges may be modified as needed by the central DBA.

Additional Privileges (local DBA)

The *RACF Group* for the local DBA will have limited authority on the application database objects created by the central DBA in the Validation DB2 subsystem. With this authority, the local DBA will have the ability to perform operational activities on the database as needed. The local DBA does not have authority to create, alter, or drop database objects in the Validation environment. The local DBA will have RACF CONNECT authority to connect userids to the Application RACF groups.

4.1.2.4 Application Plans

The central DBA will create the application plans in DB2 which can be used to associate application programs (DB2 packages). Plans will be created for the online and batch environments and will be named *xxx*C00n and *xxx*B00n respectively (where *xxx* is the assigned application identifier and *n* is a specific number). Plans will contain a generic package list as part of their definitions which will include all packages created under the collection id (*xxx*00n) assigned for the application.

The following authorities will be assigned via RACF to each plan.

Online	EXECUTE ON PLAN *xxx*C00n TO USR$*xxx*
Batch	EXECUTE ON PLAN *xxx*B00n TO DEV$*xxx*

4.1.2.5 Application Packages

In the Validation environment, Application Developers are responsible for initiating CA-Endevor processes to bind all DB2 application packages as programs are migrated from the development environment.

DB2 application package binds are executed automatically whenever a DB2 application program is migrated to the validation environment using CA-Endevor. Endevor processors are in place to associate the application currently being prepared with the appropriate DB2 collection id (*xxx*00n).

4.1.2.6 General Authorizations

All Project DBAs in the Validation DB2 subsystems will have SELECT authority on most DB2 catalog tables.

4.1.3 Production

In the DB2 Production environment, security is administered at three levels (see *DBA Roles and Responsibilities* for more details). They are:

- Central DBA
- Local DBA
- Application User

Migration of DB2 database objects from the DB2 validation environment to the production environment is the responsibility of the *central DBA*. The central DBA will perform several steps to prepare the application database in the DB2 production subsystem. These activities include:

- migrate all DB2 database objects, (database, tablespaces, tables, indexes, etc) from the validation environment;
- create online and/or batch application plans;

In the production environment, the *local DBA* is responsible for monitoring operational issues regarding the application database objects. In this environment, the local DBA will be the first point of contact in the event that a database related problem has been identified with the application. Because of this responsibility, the local DBA will have limited authority in the production DB2 subsystem.
For most host-based applications (COBOL) developed at CMS, *Application Users* will not need to have specific privileges permitting them to access DB2 tables directly.

For the most part, this group will be granted permission to execute DB2 applications only. For Application Users who must access DB2 using Dynamic SQL, (QMF, SAS, Visual Basic), additional privileges must be granted. The central DBA will grant these additional authorizations as required.

Each of the tasks noted above are described in the sections that follow.

4.1.3.1 RACF Groups

In the Production environment, the central DBA will use the Common Production RACF Group to create and maintain all application database objects. The RACF groups listed below for the *local DBA* and the *Application User* will have limited authority in the Production environment.

Production RACF Groups

xxxPRODU	*APCSS production batch user ID.*
DBA$DBss	*Common Production RACF Group Owner*. This group is used to create all DB2 objects in the production environment. (ss=subsystem ID)
DBA$*xxx*	RACF Group to be used by the *local DBA* to execute monitor DB2 objects.
USR$*xxx*P	RACF Group to be used by the *Application User* in the **production** environment. This RACF Group should facilitate user testing activities.
P*xxx*USRG	RACF Group for **production** application role-based access groups. It is controlled by EUA and may be used in place of USR$*xxx* .

where xxx is the assigned application identifier

Central DBA will add or remove individual TSO IDs to and from the production *Application User* RACF Group(s) as requested by the identified data custodian for the application data. See RACF Groups in the *Security (Development)* section of this document for details on how *Project DBA* RACF groups are assigned.

4.1.3.2 Application Database

The central DBA will create all database objects in the DB2 Production environment for the migrating application. All objects created in this environment will have an associated creator id of DBA$DBss where ss = the DB2 subsystem ID - i.e. 2P, 2W etc. The local DBA will not have the authority to create, alter, or drop any database objects in the production environment.

Preliminary Privileges (central DBA)

As part of the initial steps in setting up a DB2 application database in the Production environment, the central DBA will grant the following privileges.

Local DBA Privileges

In the production environment, the *local DBA* is responsible for monitoring operational issues regarding the application database objects. In this environment, the local DBA will be the first point of contact in the event that a database related problem has been identified with the application. Because of this responsibility, the local DBA will have limited authority in the production DB2 subsystem.

After creating the application database objects in the Production environment, the central DBA will assign the *local DBA* RACF Group the following database level privileges.

DISPLAYDB	issue the DB2 -DISPLAY DATABASE command

CA-7 Privileges

Since all batch production jobs will run under the control of CA-7, the following database privileges are assigned to the APCSS production control User ID (xxxPRODU).

DISPLAYDB	issue the DB2 -DISPLAY DATABASE command
IMAGCOPY	execute the COPY utility
LOAD	execute the LOAD utility
STATS	execute the RUNSTATS utility
STARTDB	issue the DB2 -START DATABASE command
STOPDB	issue the DB2 -STOP DATABASE command
RECOVER	execute the RECOVER utility
REORG	execute the REORG utility
REPAIR	execute the REPAIR utility

4.1.3.3 Application Plans

The following authorities are assigned via RACF to each plan.

Online	EXECUTE ON PLAN *xxx* C00n TO USR$*xxx*
Batch	EXECUTE ON PLAN *xxx*B00n TO *xxx*PRODU

Where xxx is the application identifier and n is the plan number.

4.1.3.4 Application Packages

The production control group (APCSS) is responsible for binding all DB2 application packages in the production environment.

Binding an application package in the DB2 production environment is accomplished automatically as part of the production move process. APCSS will initiate program moves using CA-Endevor. Endevor processors are in place to associate the application currently being prepared with the appropriate DB2 collection id (*xxx*00n).

4.1.3.5 General Authorizations

All *local DBAs* in the Production DB2 subsystems will have SELECT authority on most DB2 catalog tables.

4.2 RACF Security Administration

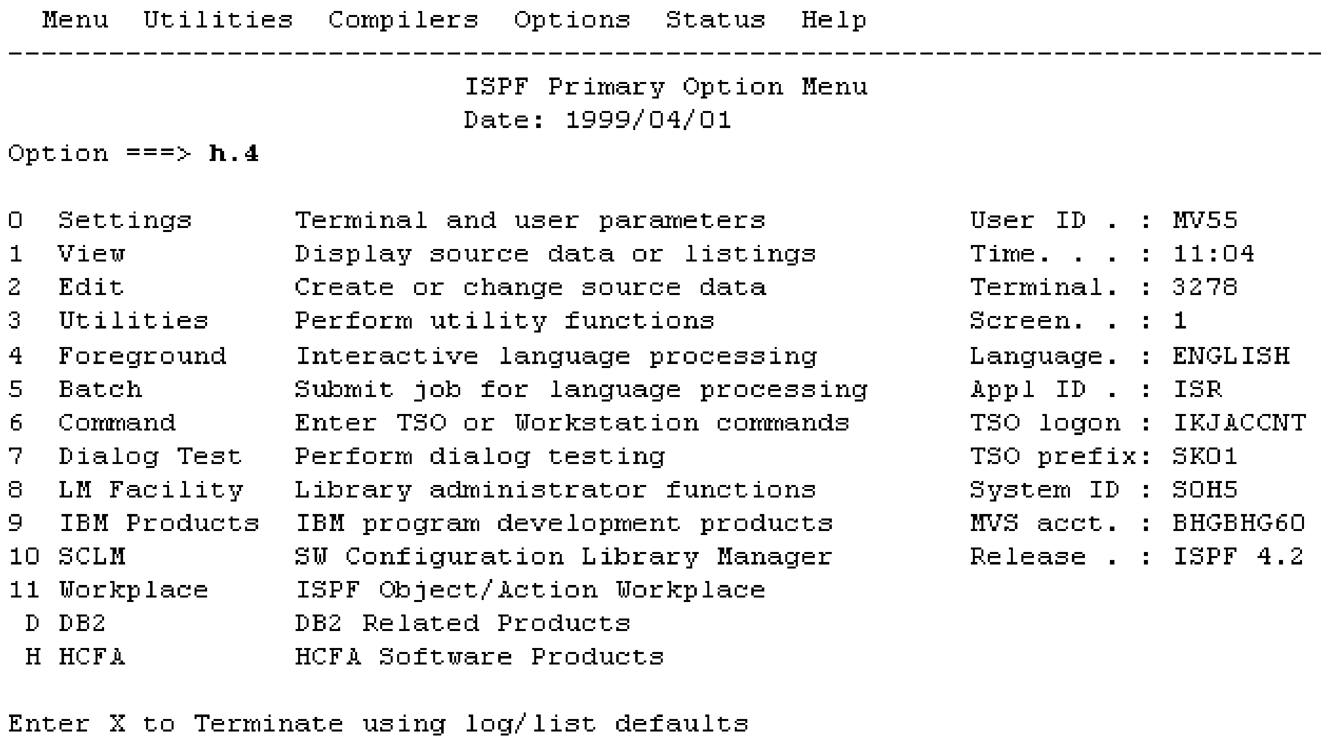
RACF (Resource Access Control Facility) is a mainframe security product that protects information and resources by controlling access to it. RACF identifies each user as they logon to the mainframe environment by means of their assigned userid. RACF administrators at CMS assign unique userids to each CMS user requiring access to mainframe resources. Each userid can be given a list of authorities permitting them to access data or issue commands. RACF administrators also have the ability to define groups (a collection of individuals) in order to administer security at a higher 'departmental' or 'functional' level.

DB2 recognizes individual userids as well as all associated RACF groups for each connected user. Given the inherent benefits of administering security at a group level (within and outside of DB2), all users of DB2 must be connected to a valid RACF group. The responsibility of associating RACF ids with RACF groups belongs to designated RACF Group Administrators. Procedures to accomplish this are listed below. For more information on this topic, contact your *Local RACF Group Administrator*.

4.2.1 Administering RACF Groups

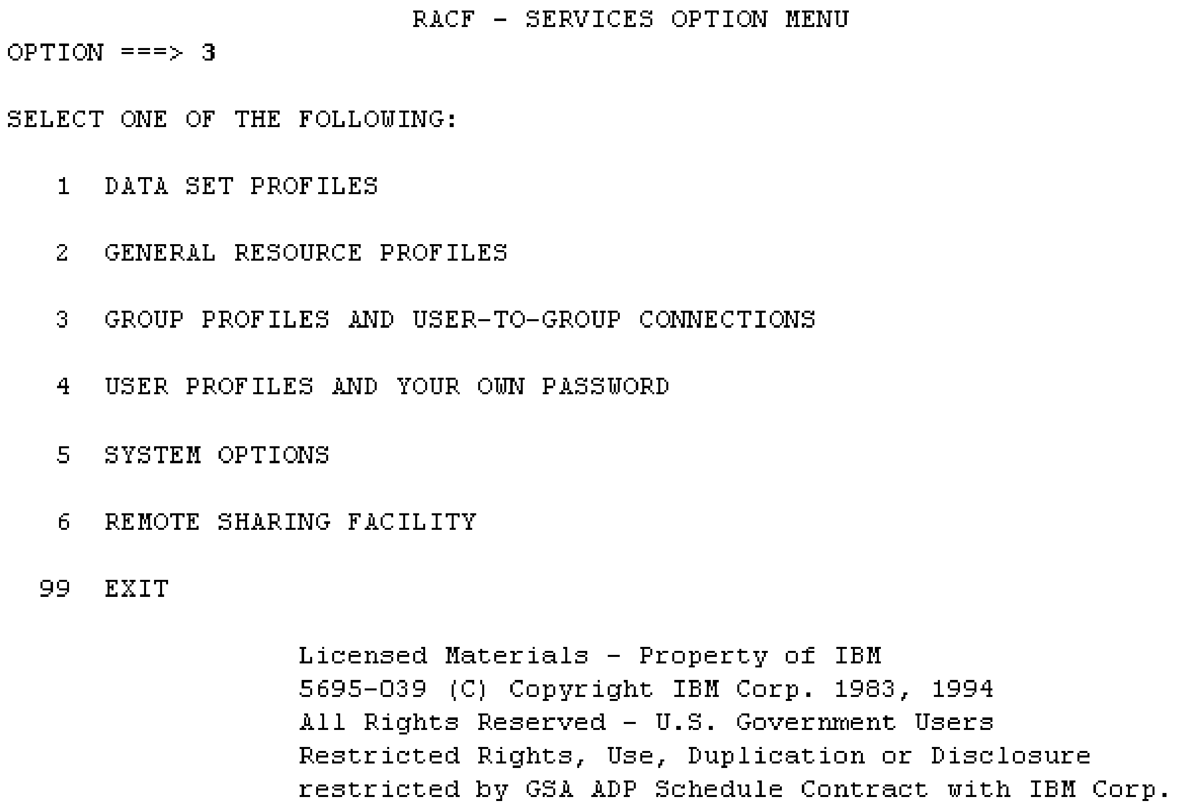
In order to use secondary auth-ids, users must be a member of the proper RACF group which is administered by the local DBA. The following is an example of how to associate individual userids to a RACF Group. Note that special RACF authority is required in order to perform these functions (see *DB2 Security - RACF Groups* for more details).

To add a user to a group, logon to TSO and access **ISPF H.4** (*CMS local menu, RACF Administration*)



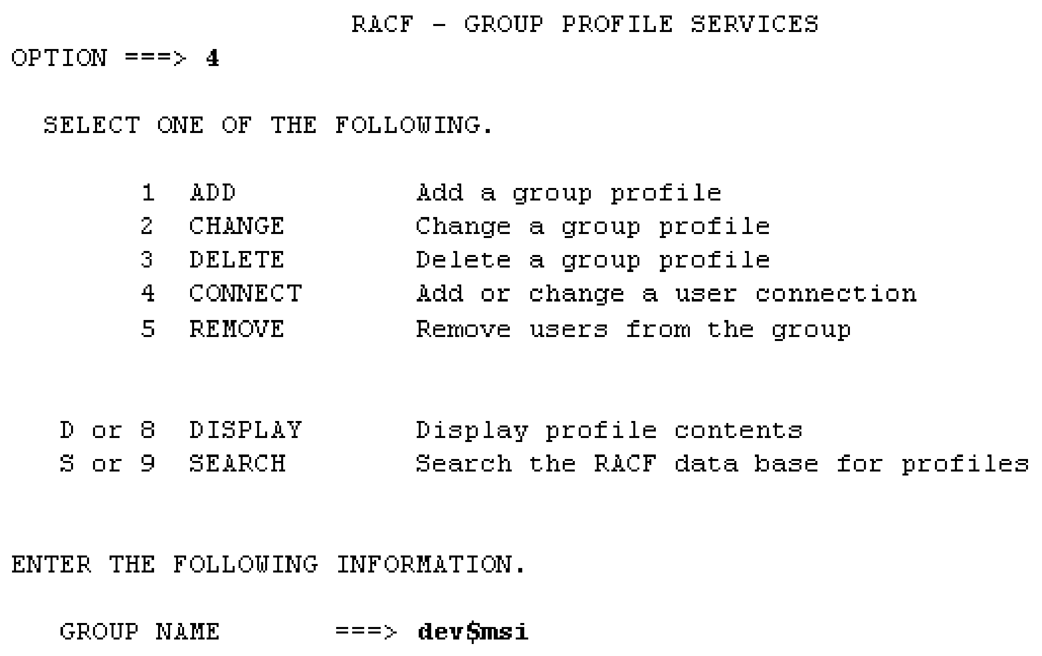
```
  Menu  Utilities  Compilers  Options  Status  Help
------------------------------------------------------------------------------
                           ISPF Primary Option Menu
                           Date: 1999/04/01
Option ===> h.4

0  Settings      Terminal and user parameters          User ID . : MV55
1  View          Display source data or listings       Time. . . : 11:04
2  Edit          Create or change source data          Terminal. : 3278
3  Utilities     Perform utility functions             Screen. . : 1
4  Foreground    Interactive language processing       Language. : ENGLISH
5  Batch         Submit job for language processing    Appl ID . : ISR
6  Command       Enter TSO or Workstation commands     TSO logon : IKJACCNT
7  Dialog Test   Perform dialog testing                TSO prefix: SK01
8  LM Facility   Library administrator functions       System ID : SOH5
9  IBM Products  IBM program development products      MVS acct. : BHGBHG60
10 SCLM          SW Configuration Library Manager      Release . : ISPF 4.2
11 Workplace     ISPF Object/Action Workplace
 D DB2           DB2 Related Products
 H HCFA          HCFA Software Products

Enter X to Terminate using log/list defaults
```

From the main RACF menu, **choose option 3** - *GROUP PROFILES AND USER-TO-GROUP CONNECTIONS.*



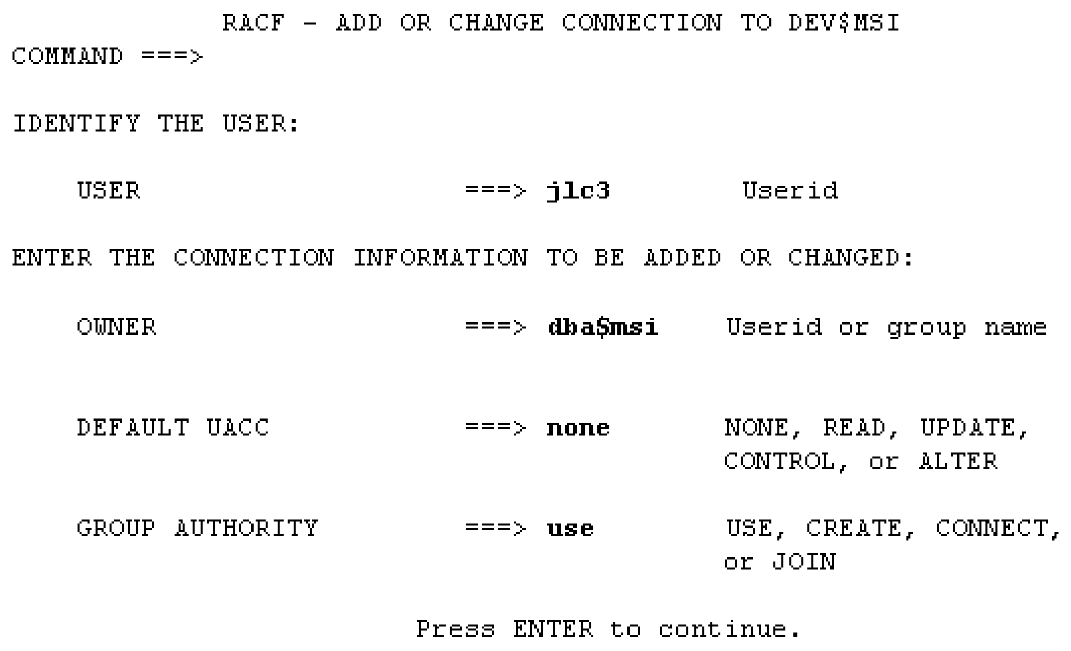
```
                       RACF - SERVICES OPTION MENU
OPTION ===> 3

SELECT ONE OF THE FOLLOWING:

   1  DATA SET PROFILES

   2  GENERAL RESOURCE PROFILES

   3  GROUP PROFILES AND USER-TO-GROUP CONNECTIONS

   4  USER PROFILES AND YOUR OWN PASSWORD

   5  SYSTEM OPTIONS

   6  REMOTE SHARING FACILITY

  99  EXIT

                 Licensed Materials - Property of IBM
                 5695-039 (C) Copyright IBM Corp. 1983, 1994
                 All Rights Reserved - U.S. Government Users
                 Restricted Rights, Use, Duplication or Disclosure
                 restricted by GSA ADP Schedule Contract with IBM Corp.
```

On the RACF Group Profile Services menu, **choose option 4** - *CONNECT* and enter the **group** (authid) name the *GROUP NAME* field.



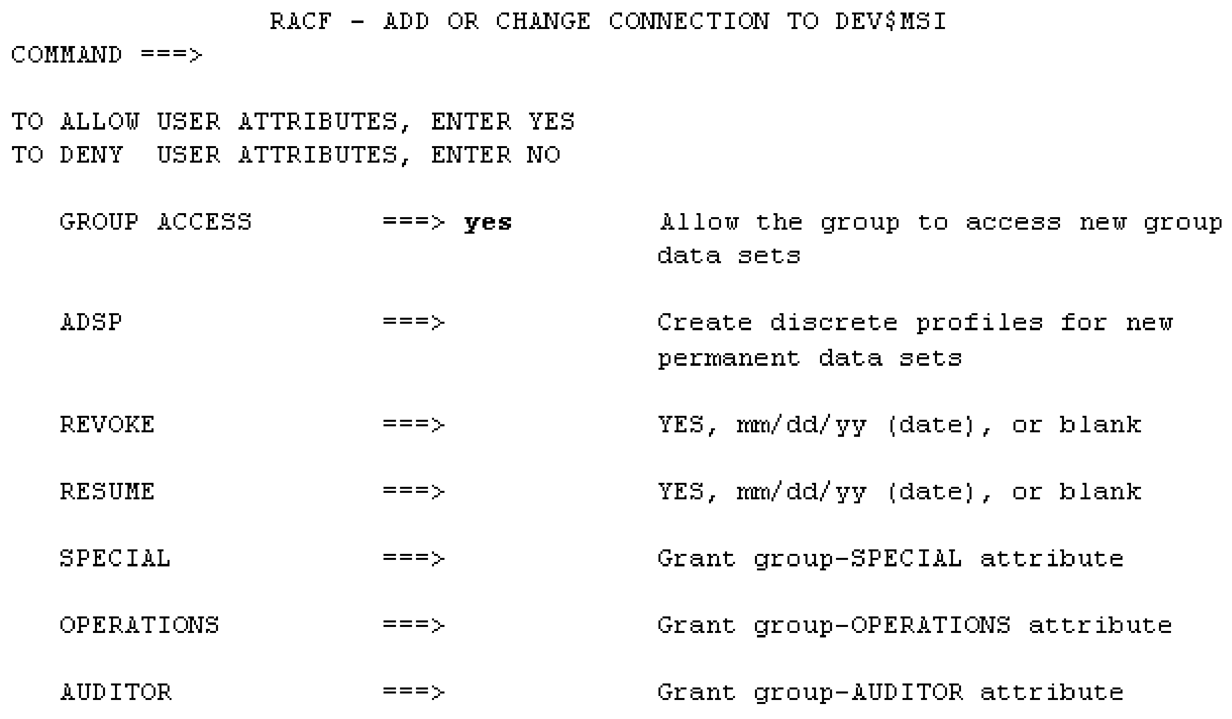
```
                     RACF - GROUP PROFILE SERVICES
OPTION ===> 4

  SELECT ONE OF THE FOLLOWING.

        1  ADD           Add a group profile
        2  CHANGE        Change a group profile
        3  DELETE        Delete a group profile
        4  CONNECT       Add or change a user connection
        5  REMOVE        Remove users from the group


   D or 8  DISPLAY       Display profile contents
   S or 9  SEARCH        Search the RACF data base for profiles


ENTER THE FOLLOWING INFORMATION.

   GROUP NAME       ===> dev$msi
```

Use the parameters below as a guide for adding the connection to the group. Substitute the individual's user-id that you want to connect.



```
             RACF - ADD OR CHANGE CONNECTION TO DEV$MSI
COMMAND ===>

IDENTIFY THE USER:

    USER                    ===> jlc3        Userid

ENTER THE CONNECTION INFORMATION TO BE ADDED OR CHANGED:

    OWNER                   ===> dba$msi    Userid or group name


    DEFAULT UACC            ===> none       NONE, READ, UPDATE,
                                            CONTROL, or ALTER

    GROUP AUTHORITY         ===> use        USE, CREATE, CONNECT,
                                            or JOIN

                          Press ENTER to continue.
```

Enter **YES** for *Group Access* and leave the other fields blank.



```
                 RACF - ADD OR CHANGE CONNECTION TO DEV$MSI
COMMAND ===>

TO ALLOW USER ATTRIBUTES, ENTER YES
TO DENY  USER ATTRIBUTES, ENTER NO

   GROUP ACCESS        ===> yes         Allow the group to access new group
                                        data sets

   ADSP                ===>             Create discrete profiles for new
                                        permanent data sets

   REVOKE              ===>             YES, mm/dd/yy (date), or blank

   RESUME              ===>             YES, mm/dd/yy (date), or blank

   SPECIAL             ===>             Grant group-SPECIAL attribute

   OPERATIONS          ===>             Grant group-OPERATIONS attribute

   AUDITOR             ===>             Grant group-AUDITOR attribute
```

4.3 Accessing DB2 Resources

This section describes how to access DB2 resources from different executing environments. Specifically, the following explains how to associate a process with a set of authorization Ids. In general, access to a DB2 subsystem at CMS will be controlled through IBM RACF using RACF groups. The sections below provide information to access DB2 resources from different executing environments. Since DB2 security rules differ from subsystem to subsystem, these instructions may not apply to every user in each of the CMS DB2 environments.

4.3.1 Dynamic SQL Applications (SPUFI, SAS, QMF, DB2 Connect, etc.)

Dynamic SQL Applications include all processes (batch and online) that present SQL statements to DB2 at execution time (see SQL Standards for more details). When DB2 receives dynamic SQL statements, it uses the value of the CURRENT SQLID special register to validate authorizations. Initially, CURRENT SQLID is set to the primary authorization ID of the individual issuing the SQL statement. The value of the primary authorization ID will differ depending on the execution environment.

The value of CURRENT SQLID can be changed to one of the RACF groups associated with the primary authorization ID using the SET CURRENT SQLID command. The following is an example of an SQL statement to change the current SQLID equal to the RACF group USR$NPSP:

```
SET CURRENT SQLID = 'USR$NPSP'
```

4.3.2 Static SQL Applications

Static SQL Applications includes all processes (batch and online) in which the SQL statements have been prepared prior to executing the application. In other words, the application was precompiled and bound to DB2 so that all authorization rules access path definitions could be determined prior to executing the applications. For these 'static' applications, DB2 generally verifies authorizations at bind time using the authorization ID of the owner of the DB2 application package or plan. The owner of the application must have authority to execute all of the SQL statements included in the application. At run time, simply DB2 verifies that the primary authorization ID of the individual running the application has the authority to execute the package or plan. The value of the primary authorization ID will differ depending on the execution environment.

4.3.3 Execution Environments

Batch DB2 applications running in a batch environment (JCL jobstream) recognize a primary authorization ID set to the RACF userid associated with the batch job. This is determined by either the USER= parameter on the job card when specified, or the TSO user id which submitted the job. In either case, it is the primary authorization ID that must have EXECUTE authority on the application plan.

CICS When executing CICS transactions, the primary ID and related secondary IDs are determined by the RACF user ID entered with the CESN signon transaction. Typically, EXECUTE authority for DB2 application packages and plans is granted to the same EUA controlled role-based access RACF group that also grants access to the CICS transaction that executes the plan .

TSO DB2 applications running in TSO recognize the RACF logon ID that was used to log on to TSO as the primary authorization ID. It is this primary authorization ID that must have EXECUTE authority on the application plan being executed in TSO.

Database Migration Procedures

This document outlines procedures required to implement a DB2 application database at Centers for Medicare & Medicaid Services (CMS). It is intended to compliment the methodology and procedures described in the *DBA Roles and Responsibilities* document for implementing relational databases.

These Application Database Migration procedures are in place to promote communication and coordination among all affected functional areas (Applications Development, Data Administration, Database Administration, Capacity Planning, Computer Operations, Production Control, Media Management, etc.). The Project GTL is responsible for coordinating the review and participation of all participants listed. The procedures are designed to ensure the success of the application development efforts as well as the integrity of the overall database architecture by proactively identifying items of concern prior to production implementation. The end result of adherence to these procedures can be a significant reduction of time and effort required to implement DB2 applications.

The following procedures identify critical points within the application and physical database development process. Each milestone is substantiated by a formal review. Because this document focuses on the physical implementation of DB2 databases at CMS, the identified review points begin after the conceptual and logical design for an application has been completed and approved.

5.1 DB2 Database Migration Overview

DB2 database objects designed to support applications developed at CMS will follow a set migration path through the three DB2 subsystem environments (development, validation, and production). Physical database objects will initially be implemented in the development DB2 subsystem. All application development activities, including unit and integration testing, will be accomplished in this environment (note: Some applications will use a separate DB2 integration environment for testing as well). When it has been determined that an application is ready for production implementation, all of the application components, including it's corresponding database objects, first will be migrated to the validation DB2 subsystem. Because the Validation and Production DB2 subsystem configurations are similar, final pre-production verification of the application (including security rules, performance, etc.) can be accomplished in the Validation environment. Once applications are verified in Validation, they can be migrated to Production for final implementation.

5.2 Preliminary Physical Database Design Review

Prior to implementing DB2 database objects in the Development DB2 subsystem, a Preliminary Physical Database Design review must take place. This review not only provides an opportunity to review and verify the application database architecture, but also serves as a means to coordinate scheduling and resource requirements.

Time frame:	Immediately preceding implementation of physical database into the development environment. [2]
Participants:	Application Development [3] , Local and Central DBA, Local and Central DA, Project GTL or Technical Lead
Purpose:	• Review the physical database design (physical and logical database model, DDL, preliminary space estimates for all database environments). • Discuss application architecture and execution environments, data flows, transaction throughput estimates, number of users and their locations, etc. • Review SQL representative of frequently executed and/or more complex queries. • Discuss application project plan, preliminary application migration strategies & time frames as well as impact on existing target environment.
Input:	• Database models: physical and approved logical • DDL for all objects to be created (tables, views, indexes, tablespaces, etc.) • Application estimates including data volumes, transaction throughput, expected usage patterns, etc. • Diagrams or documentation depicting interaction with other applications • Sample SQL for more complicated and frequently executed queries • Current Project Plan • **All documentation must be provided, via soft copy, to the Central DBA Group at least 2 weeks prior to the Preliminary Physical Database Design review meeting.**
Output:	• Approved preliminary physical database design. • Implementation schedule for development environment • Request for additional DASD (if necessary). • Modified Project Plan to include all necessary migration steps
Criteria:	• Physical database design which conforms to design standards noted in the *DB2 Standards and Guidelines* document. • Consensus from all groups that the physical design as presented can be implemented.

Comments:

[1] Assumptions: Conceptual/Logical Review process completed and approved.

[2] Actual application coding efforts ***should not*** begin until the preliminary physical database design is approved.

[3] Application Development includes representatives from the application development group responsible for the new/changed application as well as representatives from any application that is in some way affected by the new/changed application.

5.2.1 Pre-Development Migration Review Checklist

Documentation for Pre-Development walkthrough should be presented to the Central DBA Group at least ten business days before the scheduled walkthrough date. **If this date is missed your project walkthrough date will be delayed**.

Documentation due date: MM/DD/YYYY

It is the responsibility of the project GTL to invite the Central DA, Local DA, Local DBA, RACF representative and the ENDEVOR representative.

The following topics are mandatory sections that should be included in your documentation:

- Introduction
- Project plan
- Database models: physical and **approved logical**
- DDL for all Database ojects to be created
- Architecture (Diagrams depicting interaction with other applications)
- Security requirements (Master ID's/RACF requirements/DB2 Alias usage)
- Expected DASD requirements, transaction throughput
- Sample SQL of the complicated and most frequently executing queries
- Plans for Validation migration (timeline)
- Contact Information

5.3 Pre-Validation Migration Review

This review must be conducted prior to migrating database objects to validation. Database objects are moved to validation when it has been determined that the corresponding application is ready for production implementation. Due to restricted database authorities, the ***central DBA*** will be responsible for creating the physical objects (databases, tablespaces, tables, indexes, etc.) in the validation environment using a Worklist created by the Local DBA. Actual data migration will be the responsibility of the ***local DBA*** for the application. Migration of all other components for the application (source code, copybooks, datasets, etc.) will be the responsibility of the ***Application Development*** team.

Time frame:	Prior to migration of application programs and database objects to validation environment [1]
Participants:	Application Development, Local and Central DBA, Production Control, Systems Operations, Security Administration, Endevor
Purpose:	• Review the physical database design to be implemented, focusing on any changes made to the design since the preliminary database design review. • Review results of the database application architecture review. • Review database performance tests results from development environment (if any) and review database performance testing criteria for the validation environment. • Review security requirements and service level agreements for validation and production environments. • Review current space requirements for validation and production environments • Review database utility job streams (Image Copy, Recover, Reorg, Runstat…) • Discuss recovery/synchronization plans and restartability. • Discuss archive strategy. • Discuss disaster/recovery plans. • Review Preliminary Migration Plan and make recommendations for modifications where appropriate • Discuss application migration strategies & time frames, impact on existing target environment, and backout strategies.
Input:	• Current physical model • DDL for all objects to be created (tables, views, indexes, tablespaces, etc) • Documentation from *Database Application Architecture Review* (Explain Reports) • Documented results of database performance tests in development environment (when applicable) • Database performance test plan for validation environment

	• List of user classifications with required authority levels • Current space estimates for validation and production environments • Database utility job streams • Preliminary Migration Plan (provided by the Local DBA in conjunction with the Central DBA). See Sample Migration Plan • Current Project Plan • **All documentation must be provided, via soft copy, to the Central DBA Group at least 2 weeks prior to the Pre-Validation Migration Review meeting.**
Output:	• Approved physical database design • Accepted database performance test plan for validation environment • Approved Migration Plan with detailed backout strategy • Request for additional DASD for validation and/or production (if necessary). • Documented security requirements including RACF groups and associated authorizations for validation and production environments. • Modified Project Plan
Criteria:	• Approval by the central DBA staff that the physical database design as presented can be implemented.

Comments:

[1] Physical database changes identified after an application is moved to validation must first be implemented in the test environment. These changes are then subject to a Pre-Validation review prior to migration to the validation environment

5.3.1 Pre-Validation Migration Review Checklist

Documentation for Pre-Validation walkthrough should be presented to the Central DBA Group at least ten business days before the scheduled walkthrough date. **If this date is missed your project walkthrough date will be delayed**.

Documentation due date: MM/DD/YYYY

It is the responsibility of the project GTL to invite the lead application developers, Local and Central DA, business owner, Production Control, Security Administration, Endevor.

The following topics are mandatory sections that should be included in your documentation:

- Introduction
- Project plan
- Physical Database model, DDL for all objects being created
- System architecture
- Performance test results from development/integration test results, test plans for validation environment
- Security requirements (RACF requirements/list of user classifications with required authority levels)
- Space requirements for validation, expected space requirements for production environment(s)
- Database backup/recovery plans, resynchronization procedure (if dependent on other projects)
- Disaster/recovery schema
- Archiving schema
- Explain reports (representation of the busiest/most important transactions)
- Database utility job streams
- Pre-Validation migration plan with back out contingencies
- Contact Information

5.4 Pre-Production Migration Review

This review serves as the final review point prior to moving a DB2 application to production. It is primarily intended to provide an opportunity to coordinate the efforts of all affected functional areas (Applications Development, Database Administration, Capacity Planning, Systems Operations, Production Control, etc.). All DB2 applications presented for Pre-Production Migration Review must first migrate to the DB2 Validation environment.

Time frame:	Prior to migration of application programs and database objects to production environment and after application has been verified in validation environment.
Participants:	Application Development, Local and Central DBA, Systems Operations, Capacity Management, Security Administration, Production Control, Endevor
Purpose:	• Review database performance test results from validation environment. • Review security requirements and service level agreements for production environment. • Verify space requirements for production environment. • Review batch utility job streams and job scheduling requirements. • Discuss application migration strategies & time frames, impact on existing target environment, and backout strategies. • Discuss recovery/synchronization plans and restartability. • Discuss archive strategy. • Discuss disaster/recovery plans.
Input:	• Stress test results from validation environment • Current space estimates for production environment and verification that required DASD is available should be approved by the Central DBA. • Documented security requirements including RACF groups and associated authorizations for production environment (from Pre-Validation Review) with sign-off from data custodian • Documented service level agreements • Database utility job streams (Image Copy, Recover, Reorg, Runstat…) and scheduling requirements • Current project plan • Preliminary migration plan (provided by Local DBA in conjunction with the Central DBA) • **All documentation must be provided, via soft copy, to**

	the Central DBA Group at least 2 weeks prior to the Pre-Production Migration Review meeting.
Output:	• Approval for the application and database objects to move to production • Approved Migration Plan (with detailed backout strategy) for application programs and database objects • Approved security requirements including RACF groups and associated authorizations for production environment.
Criteria:	• Approval by the central DBA staff that the physical design as presented can be implemented

5.4.1 Pre-Production Migration Review Checklist

Documentation for Pre-Production walkthrough should be presented to the Central DBA Group at least ten business days before the scheduled walkthrough date. **If this date is missed your project walkthrough date will be delayed**.

Documentation due date: MM/DD/YYYY

It is the responsibility of the project GTL to invite the lead application developers, Local and Central DA, business owner, Production Control, Security Administration, Endevor.

The following topics are mandatory sections that should be included in your documentation:

- Introduction
- Project plan
- Physical Database model, DDL for all objects being created
- System architecture
- Documented service level agreements
- Stress/performance test results from validation
- Security requirements (RACF requirements/list of user classifications with required authority levels) with sign-off from data custodian
- Space requirements for production environment(s)
- Database backup/recovery procedure(s), resynchronization procedure (if applicable)
- Disaster/recovery procedure(s)
- Archiving procedure(s)
- Explain reports (representation of the busiest/most resource intensive transactions)
- Database utility job streams (production ready)
- Pre-Production migration plan with back out contingencies
- Contact Information

Database Utilities

A full suite of DB2 utilities are available at CMS that provide for overall administration of DB2 database objects. Primarily, these utilities are available to Corporate and Project DBAs. Project DBAs may, if appropriate, grant access to a limited number of these utilities to application developers as well.
CMS supports DB2 utilities from two primary vendors, IBM and BMC Software. Many products from both vendors overlap in functionality, however, there is a considerable difference in the performance of each. Therefore, CMS has established a standard set of utilities (with vendor designation) that must be adhered to when creating DB2 utility jobs intended for production. The following documents these standards.

6.1 CMS Standard DB2 Utilities

The following matrix indicates the CMS's vendor selection for DB2 utilities. All DB2 utility jobs intended for migration to the CMS production environment must be developed using these designated tools. Please refer to the appropriate vendor reference manual for specific execution procedures.

Utility	Vendor	Product
Check Data	IBM	CHECK DATA*
Imagecopy	BMC	Copy Plus
Load	BMC	Load Plus
Modify	IBM	MODIFY*
Quiesce	IBM	QUIESCE*
Rebuild Index	IBM	REBUILD INDEX*
Recover	BMC	Recover Plus
Reorg	BMC	Reorg Plus
Runstats	BMC	BMC Stats
Unload	BMC	Unload Plus

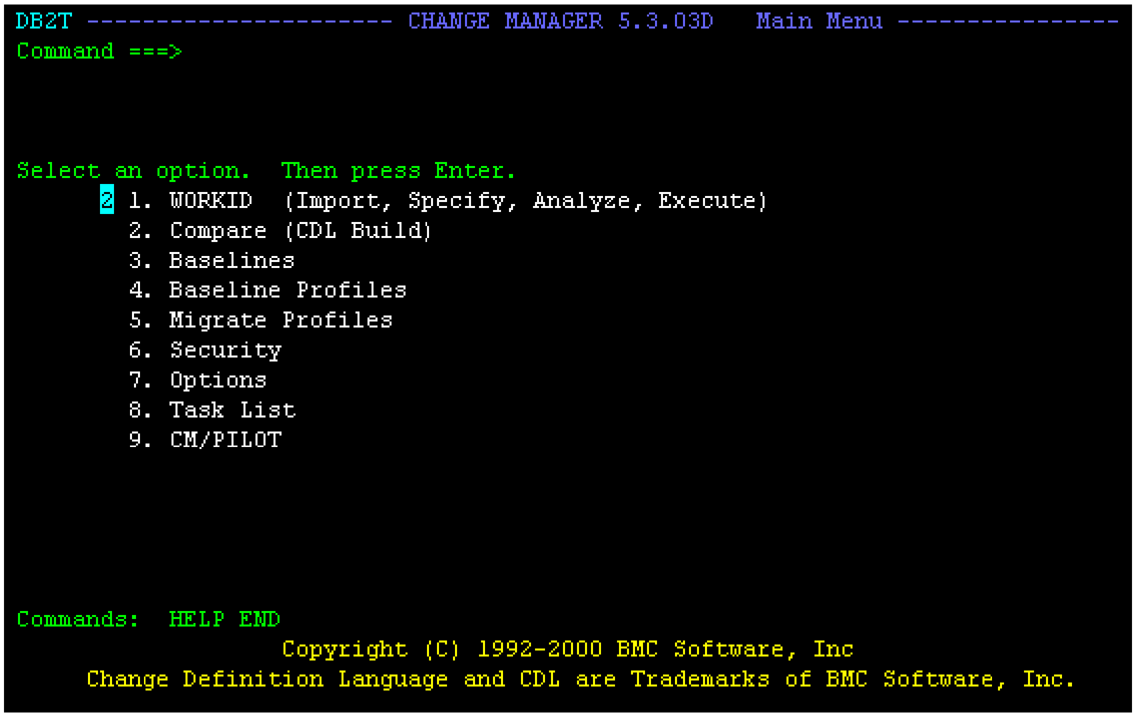
* These utilities are initiated from the IBM utility driver program DSNUTILB

6.2 How to Guides for BMC utilities

6.2.1 Compare DDL to DB2 Catalog

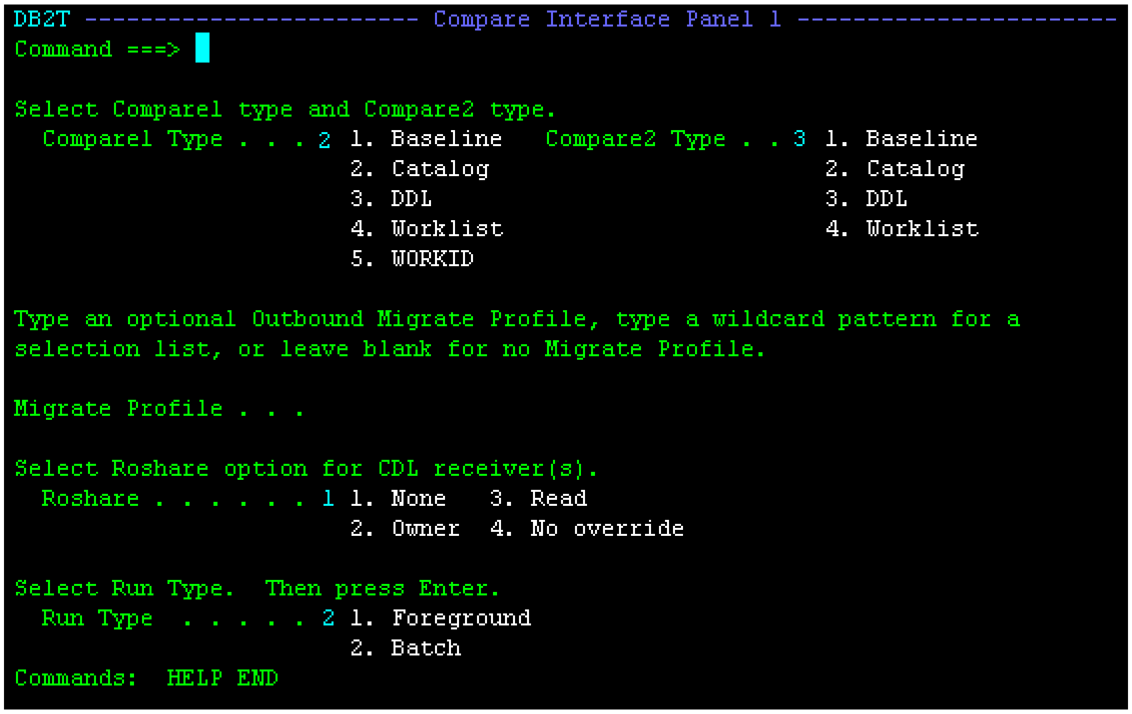
BMC Change Manager (CM) includes a feature that will compare database object descriptions from a DDL script to actual database structures on a DB2 subsystem. It is this feature that can be used to determine what database changes should be incorporated to DB2 as a result of modifications to a physical schema in ERwin.

To use the feature choose option 2 (*Compare*) from the Change Manager Main Menu.



```
DB2T ---------------------- CHANGE MANAGER 5.3.03D   Main Menu ----------------
Command ===>



Select an option.  Then press Enter.
      2 1. WORKID  (Import, Specify, Analyze, Execute)
        2. Compare (CDL Build)
        3. Baselines
        4. Baseline Profiles
        5. Migrate Profiles
        6. Security
        7. Options
        8. Task List
        9. CM/PILOT





Commands:  HELP END
                   Copyright (C) 1992-2000 BMC Software, Inc
     Change Definition Language and CDL are Trademarks of BMC Software, Inc.
```

Enter Compare1 Type "2" (*Catalog*) and Compare Type "3" (*DDL*) on the *Compare Interface* panel and press enter.

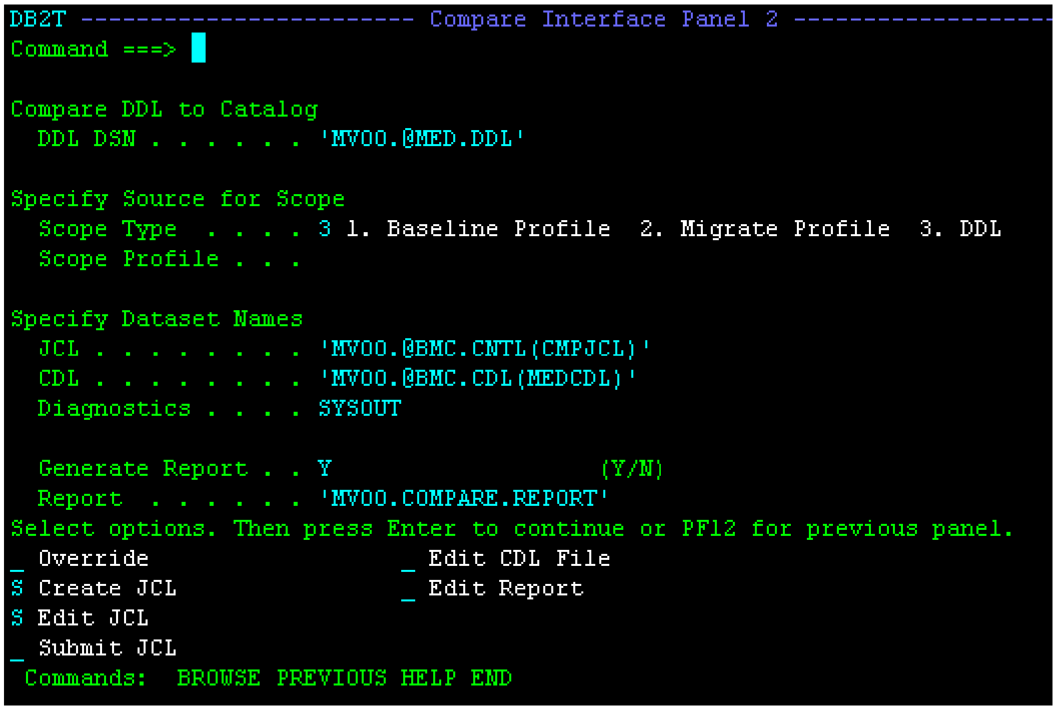

```
DB2T ----------------------- Compare Interface Panel 1 ----------------------
Command ===>

Select Compare1 type and Compare2 type.
  Compare1 Type . . . 2 1. Baseline   Compare2 Type . . 3 1. Baseline
                        2. Catalog                        2. Catalog
                        3. DDL                            3. DDL
                        4. Worklist                       4. Worklist
                        5. WORKID

Type an optional Outbound Migrate Profile, type a wildcard pattern for a
selection list, or leave blank for no Migrate Profile.

Migrate Profile . . .

Select Roshare option for CDL receiver(s).
  Roshare . . . . . . . 1 1. None    3. Read
                          2. Owner   4. No override

Select Run Type.  Then press Enter.
  Run Type  . . . . . . 2 1. Foreground
                          2. Batch
Commands:  HELP END
```

On the next panel, *Compare Interface Panel 2*, specify the name of the dataset which contains the DDL uploaded from ERwin.

Indicate the *Source for Scope.* The value you specify in this field will direct CM as to which objects it should inspect to detect differences. If you select 1 (*Baseline Profile)* or 2 (*Migrate Profile*), CM will look to the scope rules specified in either of these profile types to determine which objects to inspect. Specifying either of these types will require that you create a Baseline or Migrate profile before using the compare option. You must also specify the name of the profile in the *Scope Profile* field.

If you select Scope Type 3 (*DDL*), CM will compare the objects referenced in the DDL script against the catalog. This method is generally the correct approach if database changes are always initiated from ERwin. In our example, we assume this is the case.

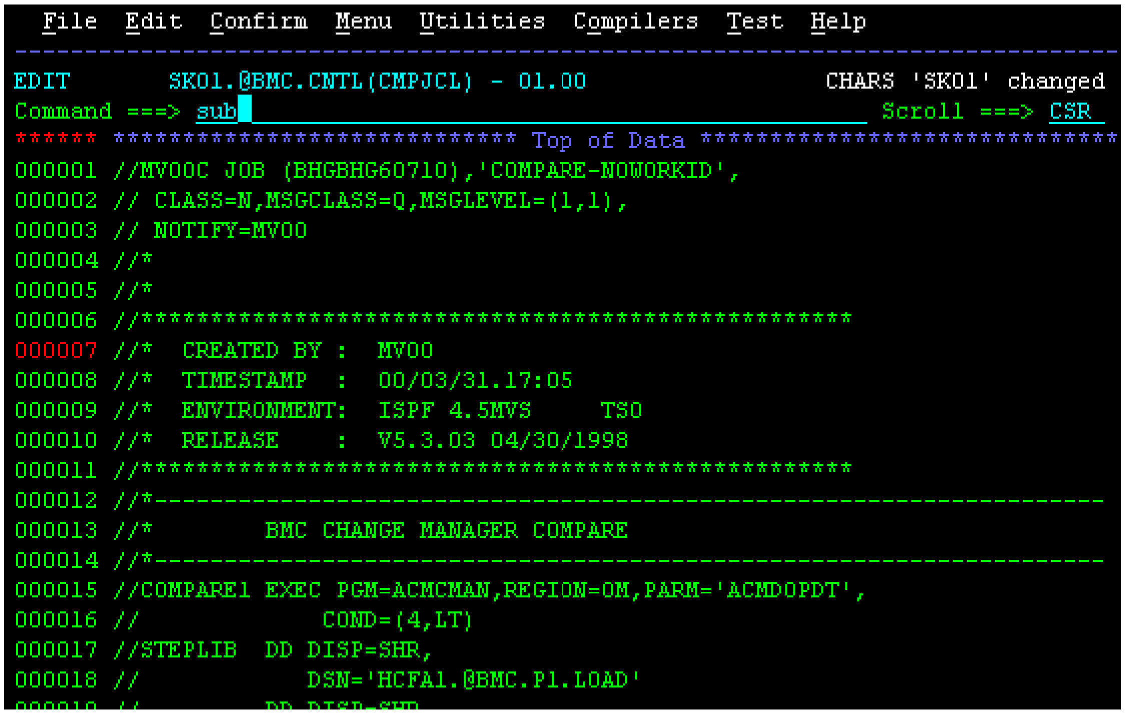
From the *Compare Interface Panel 2,* also provide the dataset name where the JCL to execute the compare process should be written along with the name of the dataset where CM should write the CDL (CM's description of identified changes). Optionally, also indicate whether a change report should be generated and if so, to what dataset. Place an "S" in the *Create JCL* and *Edit JCL* fields and press enter.



```
DB2T ------------------------ Compare Interface Panel 2 -------------------
Command ===>

Compare DDL to Catalog
  DDL DSN . . . . . . . 'MV00.@MED.DDL'

Specify Source for Scope
  Scope Type  . . . . . 3 1. Baseline Profile  2. Migrate Profile  3. DDL
  Scope Profile . . .

Specify Dataset Names
  JCL . . . . . . . . . 'MV00.@BMC.CNTL(CMPJCL)'
  CDL . . . . . . . . . 'MV00.@BMC.CDL(MEDCDL)'
  Diagnostics . . . . . SYSOUT

  Generate Report . . Y                    (Y/N)
  Report  . . . . . . . 'MV00.COMPARE.REPORT'
Select options. Then press Enter to continue or PF12 for previous panel.
_ Override                    _ Edit CDL File
S Create JCL                  _ Edit Report
S Edit JCL
_ Submit JCL
 Commands:  BROWSE PREVIOUS HELP END
```

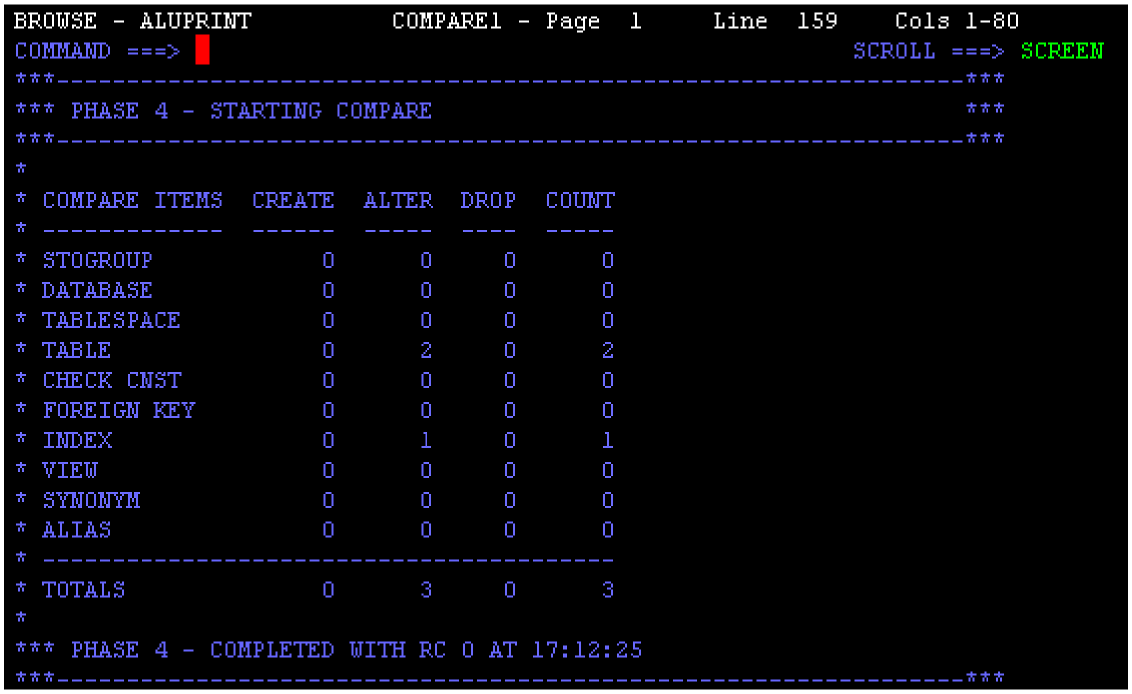
JCL to execute the compare in batch will be generated and displayed on your screen. You will have the ability to modify this JCL, if necessary, prior to executing it. Verify that the information (especially your Jobcard) is correct and submit the job.

Note that if you entered an "S" in the *SUBMIT JCL* field of the previous panel, you can submit the job by pressing *PF3* from this panel and *enter* from the next panel.



```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
 -------------------------------------------------------------------------------
EDIT       SK01.@BMC.CNTL(CMPJCL) - 01.00                 CHARS 'SK01' changed
Command ===> sub                                            Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 //MV00C JOB (BHGBHG60710),'COMPARE-NOWORKID',
000002 // CLASS=N,MSGCLASS=Q,MSGLEVEL=(1,1),
000003 // NOTIFY=MV00
000004 //*
000005 //*
000006 //***************************************************
000007 //*  CREATED BY :  MV00
000008 //*  TIMESTAMP  :  00/03/31.17:05
000009 //*  ENVIRONMENT:  ISPF 4.5MVS     TSO
000010 //*  RELEASE    :  V5.3.03 04/30/1998
000011 //***************************************************
000012 //*-----------------------------------------------------------------------
000013 //*        BMC CHANGE MANAGER COMPARE
000014 //*-----------------------------------------------------------------------
000015 //COMPARE1 EXEC PGM=ACMCMAN,REGION=0M,PARM='ACMDOPDT',
000016 //             COND=(4,LT)
000017 //STEPLIB  DD DISP=SHR,
000018 //            DSN='HCFA1.@BMC.P1.LOAD'
```

Once the job is complete, view the output in IOF and verify it ran successfully. A maximum condition code of "4" indicates a successful execution. The next two screens provide sample reports that will be generated as part of the compare process. Phase 4 of the compare provide counts of the objects that are different between the DDL and the Catalog. Phase 5 provides a count of the objects that will be created, modified, or deleted in the CDL generated from this step.

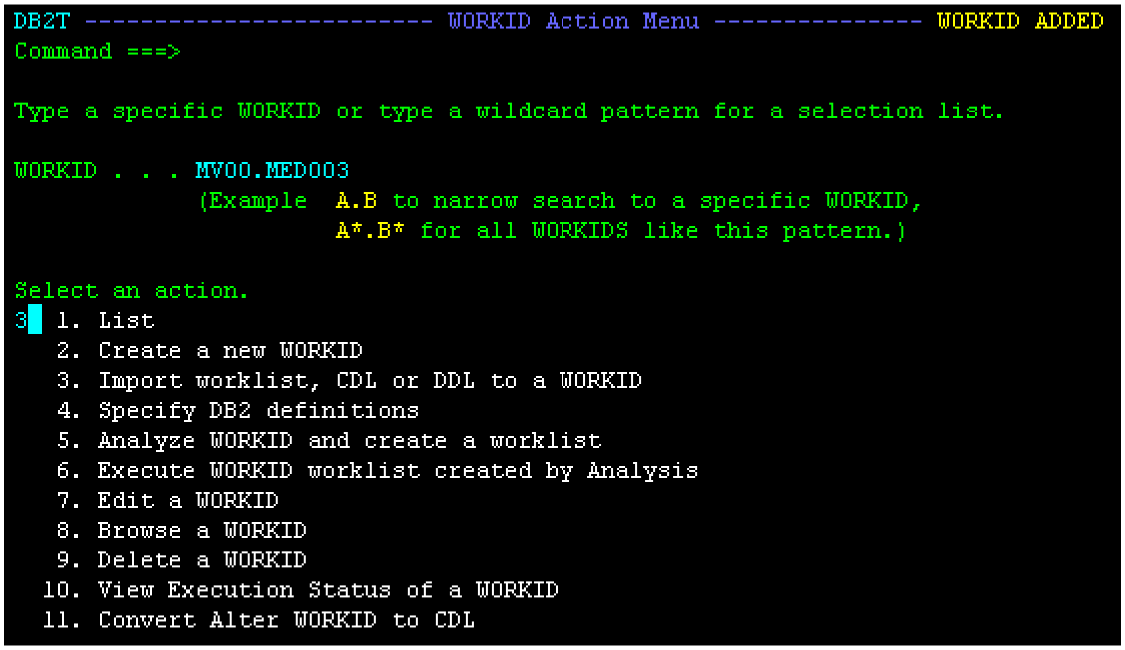


```
BROWSE - ALUPRINT           COMPARE1 - Page  1      Line  159     Cols 1-80
COMMAND ===>                                                  SCROLL ===> SCREEN
***-----------------------------------------------------------------***
*** PHASE 4 - STARTING COMPARE                                      ***
***-----------------------------------------------------------------***
*
* COMPARE ITEMS  CREATE  ALTER  DROP  COUNT
* -------------  ------  -----  ----  -----
* STOGROUP            0      0     0      0
* DATABASE            0      0     0      0
* TABLESPACE          0      0     0      0
* TABLE               0      2     0      2
* CHECK CNST          0      0     0      0
* FOREIGN KEY         0      0     0      0
* INDEX               0      1     0      1
* VIEW                0      0     0      0
* SYNONYM             0      0     0      0
* ALIAS               0      0     0      0
* -----------------------------------------
* TOTALS              0      3     0      3
*
*** PHASE 4 - COMPLETED WITH RC 0 AT 17:12:25
***-----------------------------------------------------------------***
```

Once you have reviewed the output of the compare job and are satisfied with the results, proceed to the next step, Create a Workid.

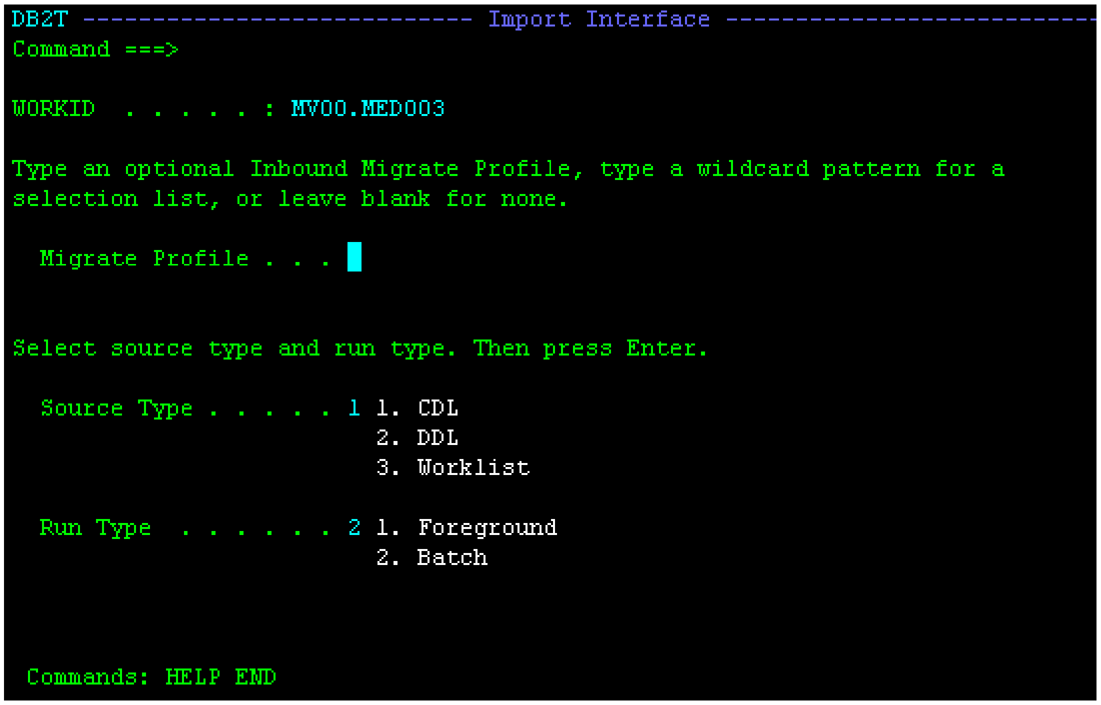
6.2.2 Import CDL to Workid

When using CDL to specify database changes in BMC Change Manager (CM), it is necessary to first *Import* the change definitions (CDL) into your workid. The Import process will read the CDL and convert it to change specifications in your workid. To use this feature, choose option 3 in the *WORKID Action Menu* and press enter.



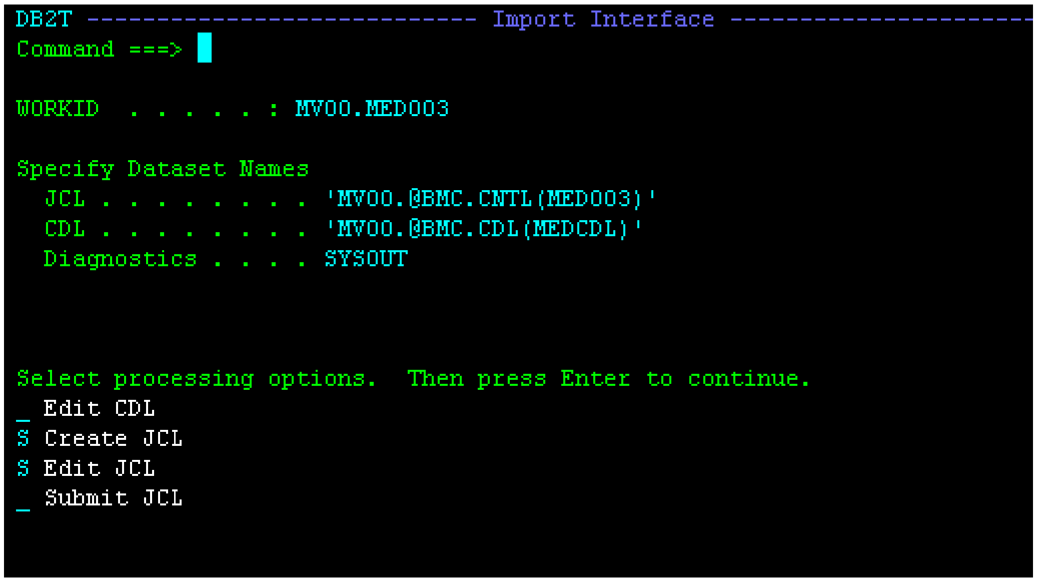
```
DB2T ------------------------- WORKID Action Menu --------------- WORKID ADDED
Command ===>

Type a specific WORKID or type a wildcard pattern for a selection list.

WORKID . . . MV00.MED003
             (Example  A.B to narrow search to a specific WORKID,
                       A*.B* for all WORKIDS like this pattern.)

Select an action.
3  1. List
   2. Create a new WORKID
   3. Import worklist, CDL or DDL to a WORKID
   4. Specify DB2 definitions
   5. Analyze WORKID and create a worklist
   6. Execute WORKID worklist created by Analysis
   7. Edit a WORKID
   8. Browse a WORKID
   9. Delete a WORKID
  10. View Execution Status of a WORKID
  11. Convert Alter WORKID to CDL
```

On the *Import Interface* menu, verify that the following options have been selected and press enter.



```
DB2T ---------------------------- Import Interface --------------------------
Command ===>

WORKID  . . . . . : MV00.MED003

Type an optional Inbound Migrate Profile, type a wildcard pattern for a
selection list, or leave blank for none.

  Migrate Profile . . .


Select source type and run type. Then press Enter.

  Source Type . . . . . 1 1. CDL
                          2. DDL
                          3. Worklist

  Run Type  . . . . . . 2 1. Foreground
                          2. Batch



 Commands: HELP END
```

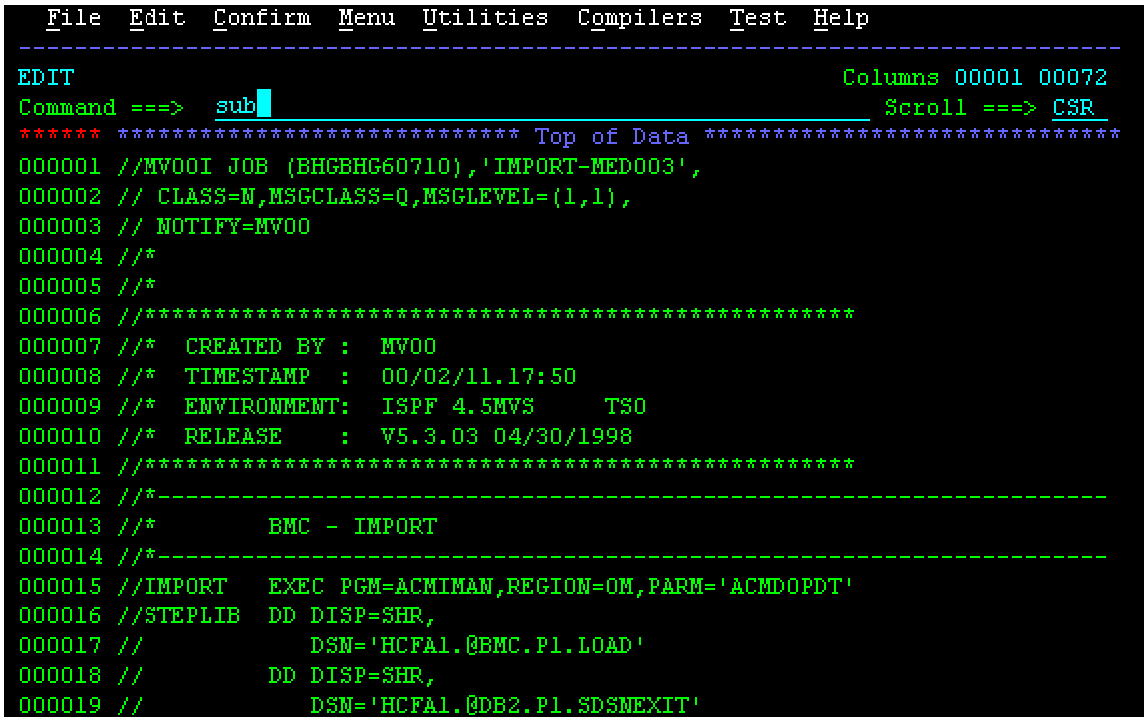
On the next *Import Interface* panel, specify the name of the dataset where JCL to execute the import process is to be written along with the name of the dataset that contains the CDL. Generally, this is the dataset name that was created in the Compare process (see *Compare DDL to DB2 Catalog* for more details). Place an "S" in the *Create JCL* and *Edit JCL* fields and press enter.



```
DB2T ---------------------------- Import Interface ----------------------
Command ===>

WORKID  . . . . . : MV00.MED003

Specify Dataset Names
  JCL . . . . . . . . 'MV00.@BMC.CNTL(MED003)'
  CDL . . . . . . . . 'MV00.@BMC.CDL(MEDCDL)'
  Diagnostics . . . . SYSOUT



Select processing options.  Then press Enter to continue.
_ Edit CDL
S Create JCL
S Edit JCL
_ Submit JCL
```

JCL to execute the import in batch will be generated and displayed on your screen. You will have the ability to modify this JCL, if necessary, prior to executing it. Verify that the information (especially your jobcard) is correct and submit the job.

Note that if you entered an "S" in the *SUBMIT JCL* field of the previous panel, you can submit the job by pressing *PF3* from this panel and *enter* from the next panel.



```
   File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
 -------------------------------------------------------------------------------
 EDIT                                                   Columns 00001 00072
 Command ===>  sub                                         Scroll ===> CSR
 ****** ***************************** Top of Data ******************************
 000001 //MV00I JOB (BHGBHG60710),'IMPORT-MED003',
 000002 // CLASS=N,MSGCLASS=Q,MSGLEVEL=(1,1),
 000003 // NOTIFY=MV00
 000004 //*
 000005 //*
 000006 //****************************************************
 000007 //*  CREATED BY :  MV00
 000008 //*  TIMESTAMP  :  00/02/11.17:50
 000009 //*  ENVIRONMENT:  ISPF 4.5MVS     TSO
 000010 //*  RELEASE    :  V5.3.03 04/30/1998
 000011 //****************************************************
 000012 //*-----------------------------------------------------------------
 000013 //*        BMC - IMPORT
 000014 //*-----------------------------------------------------------------
 000015 //IMPORT   EXEC PGM=ACMIMAN,REGION=0M,PARM='ACMDOPDT'
 000016 //STEPLIB  DD DISP=SHR,
 000017 //            DSN='HCFA1.@BMC.P1.LOAD'
 000018 //         DD DISP=SHR,
 000019 //            DSN='HCFA1.@DB2.P1.SDSNEXIT'
```

Once the job is complete, view the output in IOF and verify it ran successfully. A maximum condition code of "4" indicates a successful execution. When you have completed the import process, move on to the next step, *Analyze the Workid.*

Database Performance Monitoring

7.1 TOP 10 SQL Performance Measures

In an effort to proactively monitor the efficiency of the SQL statements executing in the production DB2 subsystems, the Central DB2 DBA staff has developed a set of reports for capturing high usage SQL statements. The reports come from the BMC APPTUNE product and will consist of one static SQL report and one dynamic SQL report per day for each production subsystem (DB2W will have only a static report). Each daily report will have the Top 10 SQL statements that used the most total CPU time.

Subsystem	**Static Report**	**Dynamic Report**
DB1P	Yes	Yes
DB2P	Yes	Yes
DB3P	Yes	Yes
DB2W	Yes	NO*

** Collecting SQL data for DB2W exhausted the available storage allocated to the APPTUNE product and had to be turned off.*

The complete reports are stored on the mainframe for 90 days and have the following GDG names:

Subsystem	**Static Report GDG Dataset Name**	**Dynamic Report GDG Dataset Name**
DB1P	P@DBA.@DB2.DB1P.TOP10.SF301	P@DBA.@DB2.DB1P.TOP10.DF301

DB2P	P@DBA.@DB2.DB2P.TOP10.SF301	P@DBA.@DB2.DB2P.TOP10.DF301
DB3P	P@DBA@DB2.DB3P.TOP10.SF301	P@DBA.@DB2.DB3P.TOP10.DF301
DB2W	P@DBA@DB2.DB2W.TOP10.SF301	Not Collected

The data from each of these reports will be stored for 90 days in a DB2 table named SYS$ADM.DBA_TOP10_PERF. The table structure is as follows:

Column Name	**Column Description**
SSID	DB2 Subsystem ID – DB1P ,DB2P, DB3P, DB2W
INTERVAL_DATE	Date that SQL executed
RPT_TYPE	Report Type – SF301 for Static, DF301 for Dynamic
OWNER	Object Owner
COLLID	Collection ID
PACKAGE	Package Name
STMTNO	SQL Statement ID
SQL_CALLS	Number of Times SQL was executed
TOT_CPU_TIME	Total CPU Time Consumed by the SQL
AVG_CPU_TIME	Average CPU Time for One Execution of the SQL
TOT_ELAP_TIME	Total Elapsed Time Consumed by the SQL
AVG_ELAP_TIME	Average Elapsed Time for One Execution of the SQL

On a weekly basis each Central DBA will be responsible for analyzing an SQL that is at the top of the Top 10 report for their respective applications and make recommendations and take actions to attempt to improve the performance of the high usage SQL. **Please note that Central DBAs can only make recommendations and do not have the authority to require an application to make a change**. A spreadsheet named the WeeklyTop10Analysis.xls will be used to track the status of this analysis and will be stored on the CMS network (H:\DB2DBA\WeeklyTop10Analysis.xls).

In addition a weekly report will be emailed to those interested providing a list of the Top 45 SQL that used the most Total CPU Time for the week.

IBM Manuals and Publications

Use the following link to access IBM's manuals and publications for their DB2 product:

http://www-4.ibm.com/software/data/db2/os390/library.html

Note: This is an external link that is not the responsibility of, or under the control of, the Centers for Medicare & Medicaid Services (CMS). CMS does not endorse any products, services or web sites.

Glossary

Glossary of Terms

active log
The portion of the DB2 log to which log records are written as they are generated. The active log always contains the most recent log records, whereas the archive log contains those records that are older and will no longer fit on the active log.

alias
An alternate name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 subsystem.

application plan
A control structure used by DB2 to process SQL statements during the execution of an application. The application plan is created during the BIND process.

application package
An object containing a set of SQL statements that have been bound statically and that are available for processing.

archive log
The portion of the DB2 log that contains log records that have been moved (off-loaded) from the active log because there is no more space for them in the active log.

BIND
The DB2 statement that creates an application plan or package. A DB2 bind is to SQL what a compile and link edit is to host language source code. The BIND process determines which access paths to the data the application plan will utilize. BINDs can be done automatically and dynamically.

BSDS (Boot Strap DataSet)
A VSAM KSDS dataset that contains name and status information for DB2, as well as relative byte address (see RBA below) range specifications for all active and archive log datasets. It also contains passwords for the DB2 Directory and Catalog, and lists of conditional restart and checkpoint records.

bufferpool
The main storage reserved to satisfy the buffering requirements for one or more table- spaces or indexes.

CESN

The kind of sign-on transaction used in CICS during which the User ID is entered.

CICS (Customer Information Control System)
One of three (TSO, CICS, IMS) MVS host environments for which DB2 provides services to manage the interface between the host address space and DB2 address space.

clustering index
The index that determines how rows are physically ordered in a tablespace.

collection
An ID or qualifier which is assigned to DB2 packages. Used to manage packages in logical groups.

column
The equivalent of a conventional data field that are the attributes of rows.

COMMIT
A SQL statement that terminates a unit of recovery. A COMMIT releases all locks. Data that was changed is made permanent within the database.

commit point
A point in time when data is considered consistent. See ***point-of-consistency*** below.

constraint
A rule that limits the values that can be inserted, deleted, or updated in a table.

DASD (Direct Access Storage Device)
The physical volume where data is stored.

database
A collection of DB2 objects. When you define a database, you give a name to an eventual collection of tables, indexes, and tablespaces in which they reside. A single database, for example, may contain all the data associated with beneficiaries (names, provider ID, Medicaid classification, etc.). Databases do not physically exist but are a logical group of DB2 objects that DB2 uses to assign certain authorities and that permit sensible management of data.

DB2 Catalog
DB2-maintained tables that contain descriptions of DB2 objects such as tables, views, indexes, and authorizations.

DB2 Directory
The system database that contains internal objects such as database descriptors, skeleton cursor tables, and opening/closing log of relative byte addresses (RBA) for tablespaces.

DB2 command
An instruction to the DB2 subsystem allowing a user to start or stop DB2, to display information on current users, to start or stop databases, to display information on the status of databases and so on. These commands are generally entered using the DSN Command Processor in DB2 Interactive (DB2I).

DB2 Interactive (DB2I)
The DB2 facility that provides for the execution of SQL statements, DB2 (operator) commands, programmer commands and utility invocation.

DB2 Objects
DB2 data objects are databases, storage groups, tablespaces, tables, indexes, indexspaces and views. A more detailed description of these objects can be found in the *IBM DB2 Administration Guide*. In general, an object is anything you can create or manipulate with SQL.

DB2 utility
A standard MVS batch job which requires that DB2 be running. Example DB2 utilities are COPY, LOAD, QUIESCE and REORG.

DBRM (Database Request Module)
The module that contains information about SQL statements in an application program. The DBRM is created as output from the DB2's precompiler and used as input to the BIND process.

DCL (Data Control Language)
One of the three components of SQL (the others being DDL and DML) comprised of SQL statements that control authorization to access and use the database, i.e. grant and revoke DB2 privileges.

DCLGEN (Declarations Generator)
A subcomponent of DB2 which automatically generates host language declarations (e.g., COBOL copybooks) for SQL tables. In other words, the general copy library for tables.

DDL (Data Definition Language)
One of the three components of SQL (the others being DCL and DML) comprised of SQL statements that define objects that make up the databases, i.e., create, alter and delete DB2 objects.

distinct type
A user-defined data type that is internally represented as an existing type (its source type), but is considered to be a separate and incompatible type for semantic purposes.

DML (Data Manipulation Language)
One of the three components of SQL (the others being DCL and DDL) comprised of SQL statements that retrieve and update the data, i.e., select, insert, update and delete data.

DMS (Disk Management Storage)
A disk management tool used to manage DASD pools.

DRDA (Distributed Relational Database Architecture)
The kind of database architecture using the relational data model and where some or all data is stored on a computer different from the computer used by programs or users that access the data.

dynamic SQL
SQL statements are created, prepared, and executed while a program is executing. Therefore, it is possible with dynamic SQL to change the SQL statement during program execution and have many variations of a SQL statement at run time.

embedded dynamic SQL
SQL statements that are not completely composed at the time the application in which the statement is embedded is prepared. Instead, the statements are prepared and executed while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution. SQL which is coded within an application program.

embedded static SQL
Also known as *static SQL*, these are SQL statements that are created and prepared before program execution. After the SQL statement is embedded in an application, the only variance allowed at execution is in the values of the host variables.

entity
Any person, place, thing, or event about which the enterprise collects data.

foreign key
A column, or combination of columns, whose values must match those of a primary key in another table in order to maintain a relationship between tables. A given

foreign key value represents a reference from the row(s) containing it to the row containing the matching primary key value.

HDCUG
Abbreviation for *CMS Data Center Users' Guide.*

host structure
A group of host variables. A host structure is defined using host language statements.

host variable
Any data element declared in a host language (COBOL, PL/1, etc) that is referenced in an embedded SQL statement. Host variables are used to transfer data to and from DB2, evaluate a WHERE or HAVING clause, receive or assign special register values such as CURRENT DATE or CURRENT SQLID, or set or detect the existence of NULL values.

image copy
An exact reproduction of all or part of a tablespace. The DB2 COPY utility can make full image copies (to copy the entire tablespace) or incremental image copies (to copy only those pages modified since the last image copy).

index
An index is an ordered set of pointers to data in a table and is stored separately from the table. Each index is a separate physical structure based on data values in one or more columns of the table. Once an index is created, it is maintained by DB2 so that DB2 decides when to use or not to use the index to access data in the table. Indexes can be used to enforce uniqueness of rows in a table and enhance the performance of data retrieval operations.

indexspace
A page set used to physically store the entries of one index (or index partition); automatically assigned when an index is created.

key
A column or an ordered collection of columns identified in the description of a table, index, or referential constraint.

KSDS (Key Sequenced Dataset)
A type of data organization used by VSAM datasets that uses key sequences.

leaf page
The lowest-level index page which contains the keys and record IDs (RIDs) that identify individual rows in the related table.

lock
A control structure used to serialize data updates and thereby prevent access to inconsistent data during concurrent access by multiple users.

log
A collection of records that describe the events that occur during DB2 execution and their sequence. The information recorded is used for recovery in the event of a failure during DB2 execution. Each log record is identifiable by the relative byte address (RBA) of the first byte of its header. The record's RBA (see ***log RBA*** below) is a similar timestamp in that it uniquely identifies records that start at a particular point in the continuing log.

log RBA (Relative Byte Address)
The address of each byte in the DB2 log obtained by its offset from the beginning of the log.

MVS (Multiple Virtual Storage)
IBM application development platform using virtual storage where each job to be executed is assigned its own range of addresses (address space) between 0 and 16 megabytes.

nonleaf page
A page in an index structure that contains keys and page numbers of subordinate nonleaf or leaf pages. The nonleaf page never points directly to data.

null
A data state that indicates the absence of information.

object
Anything that can be created or manipulated with SQL -- that is, databases, tablespaces, tables, views, or indexes. See ***DB2 Objects*** above.

ODBC (Open Data Base Connectivity)

package
see *application package*.

page
A unit of storage within a tablespace (4K or 32K) or indexspace (4K) that is the unit of I/O. In a tablespace, a page contains one or more rows of a table.

page set

The total collection of pages that make up an entire table space or index space. Each page set is made from a collection of VSAM data sets.

plan
See ***application plan***.

point-of-consistency
A point in time at which all data is static and consistent. There are three types of point-of-consistency: 1) an application point-of-consistency guarantees the integrity of the application data at a particular point in time; 2) a tablespace set point-of-consistency refers to those points at which all of the data in a set of tablespaces is static and consistent; 3) a system point-of-consistency refers to those points in a DB2 system at which all data -- both system and user -- within that system are static and consistent.

primary key
A column, or combination of columns, within a table whose values together form the "principal unique identifier" of rows in that table. In other words, a table's primary key serves to uniquely identify each row in that table and consists of those columns required to ensure that no duplicate rows occur in the table. This non-duplication ensures that each instance of the entity (the table) is unique.

quiesce point
An established point that corresponds to a point-of-consistency for all identified tablespaces in the control statement. The value of the quiesce point (log RBA) is stored in the catalog table SYSIBM.SYSCOPY.

RACF (Resource Access Control Facility)
A security system that controls access to production resources by assigning privileges to users.

RBA (Relative Byte Address)
See ***Log RBA*** above.

RCT (Resource Control Table)
CICS System table containing parameter settings which control application connections between CICS and DB2. The RCT is used by the CICS attachment facility to govern the way in which DB2 resources are accessed. Rules specified in the RCT include the method DB2 should use to allocate application plans, check authorizations, allocate resources, etc.

RDBMS (Relational Data Base Management System)
A kind of database management system (DBMS) based on the relational data model in which the DBMS presents the complete information content of the database to the user as a collection of two-dimensional tables (columns and rows).

recovery
The process of rebuilding databases after a system or application failure.

recovery log
See ***log*** above.

referential integrity
The condition that exists when all intended references from data in one column of a table to data in another column are valid. It maintains the consistency of relationships between tables by requiring every foreign key value in a table to have a matching primary key value in a corresponding table or else be labeled null.

RID (Record IDentifier)
The internal record identifier used to locate a given row in a table. It is composed of the page number and record ID within the page.

row
The smallest unit of data in a table that can be inserted or deleted; physically stored as records on a page. It is one instance of the entity that its table represents; conventionally called the record occurrence.

schema
A logical grouping for user-defined functions, distinct types, triggers, and stored procedures. When an object of one of these types is created, it is assigned to one schema, which is determined by the name of the object. For example, the following statement creates a distinct type T in schema C:
CREATE DISTINCT TYPE C.T ...

SQL (Structured Query Language)
A database language, originally developed by IBM, to support the definition, manipulation, and control of data in a relational database.

SQLCA (SQL Communication Area)
The communication block of variables used by DB2 to inform an application program of the status of the system as a result of a prior SQL call.

SQLCODE
Variable passed from DB2 to an application program, tool, or utility which indicates the status of the most recently issued executable SQL call. SQLCODE can take on one of three values: 1) SQLCODE = 0 indicates that the last SQL command executed successfully; 2) SQLCODE >0 indicates the last SQL command executed successfully with some warnings; and 3) SQLCODE < 0 indicates that an error condition was encountered and the last SQL command did not execute successfully.

SQLDA (SQL Descriptor Area)
The group of variables used in the execution of some SQL statements. It is used in application programs containing embedded dynamic SQL.

SPUFI (SQL Processor Using File Input)
A subcomponent of DB2I that allows interactive access to data from TSO. It allows a user to execute SQL statements without embedding them in a program.

storage group
A named set of direct access storage device (DASD) volumes from which DB2 automatically allocates storage space, defines the necessary VSAM datasets, and extends or deletes them as required.

stored procedure
A user-written application program that can be invoked through the use of the SQL CALL statement.

subject area
A high-level category of data that exists in an organization. Subject areas are based on broad grouping of entities that an enterprise is concerned with in performing its work.

table
Collections of rows (also called tuples) having the same columns (attributes). It contains data about a given entity. The rows of a table are unordered and physically stored in a tablespace. For example, a beneficiary entity (table) would consist of a row for each beneficiary and columns such as beneficiary ID, beneficiary name, and beneficiary status. A table may be defined to have a primary key, a column or set of columns whose values uniquely identify each row (beneficiary ID in the beneficiary entity).

table check constraint
A user-defined constraint that specifies the values that specific columns of a base table can contain.

tablespace
A tablespace is a VSAM dataset in which one or more tables are stored. It is a physical object for which disk space is allocated using primary and secondary quantities. A tablespace is broken up into pages of four kilobytes each. Many DB2 utilities including RUNSTATS, REORG and COPY, run against tablespaces (not tables).

tablespace set

The group of all tables related to each other through referential constraints, and which must be recovered to the same point-of-consistency.

thread
The DB2 structure that defines a connection between an application and DB2 in order to control access. At any given time, the number of active threads equals the number of users (programs, utilities, interactive users, etc.) accessing DB2. If the maximum number of concurrent threads (set at installation) is exceeded, an application must wait until a thread becomes available.

trigger
A set of SQL statements that are stored in a DB2 database and executed when a certain event occurs in a DB2 table.

user-defined function (UDF)
A function that is defined to DB2 by using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A user-defined function can be an *external function, a sourced function, or an SQL function*. Contrast with *built-in function*.

view
A view is a logical table that is derived from combining tables and/or other views into a single, logical table. A view can also be a subset of columns from a single table or view. Like a table, a view consists of rows and columns, but unlike a table, the data in a view is not physically stored. A view is defined to DB2 by referencing other views or tables, and the definition of the view is the only thing that is physically stored in the DB2 Catalog. When a user references a view, DB2 assembles the data from the underlying tables and views according to the definition. It is essentially transparent to the user whether the base table or logical view is being used. Views are a powerful tool used in relational databases to simplify SQL coding and can be used as a security device to restrict which columns of a table a user can access.

value
The intersection of a column and a row which is the smallest unit of data that can be retrieved or changed.

VSAM (Virtual Storage Access Method)
A mass storage access method that contains logical rather than physical datasets.